LABINVENT

DOC TECHNIQUE - pour Développeurs (adapter, contribuer)

URL officielle de ce doc : https://tinyurl.com/labinvent-dev
Auteurs: E. Pallier
Version: 30/3/2021
⇒ *Pour faire une copie Word, Libre/Open Office, ou PDF de ce doc : menu "Fichier/Télécharger…"*

*Cette **documentation destinée aux développeurs** entre dans des détails techniques sur le logiciel LabInvent et explique comment contribuer à son évolution, comment l'adapter, le configurer, l'améliorer.*

*La **documentation technique générale** expliquant la phase d'installation et de configuration du logiciel est **ici***

*D'autre part, beaucoup **d'autres informations techniques** sont rassemblées dans la partie **Annexes**.*

⇒ ***TODO LIST*** *(Liste des correctifs bugfixes et améliorations en cours)*

Un gros effort est fait continuellement pour que cette doc soit le plus à jour et pertinente possible,

merci de bien vouloir la lire ATTENTIVEMENT.

Les auteurs de cette doc eux-mêmes la suivent à la lettre, pourquoi donc feriez-vous autrement ?

N'hésitez pas aussi à la mettre à jour vous-mêmes quand c'est nécessaire, ou bien à soumettre vos suggestions à :

epallier AT irap POINT omp POINT eu

HISTORIQUE DES MISES A JOUR DE CETTE DOCUMENTATION

- 8/2/21 Nouveau chapitre dédié à la “gestion des utilisateurs” qui regroupe tous les aspects liés à ce thème
- 20/1/21 **Ajout** de plusieurs chapitres (qui étaient inappropriés dans la doc d’installation)
- 7/12/20 Mise à jour du chapitre “Autorisations” pour préciser les 2 niveaux (action et vue)
- 25/11/20 Mise à jour du chapitre “Etiquettes” pour ajouter la nouvelle étiqueteuse Dymo LabelManager 420P
- 19/10/20 Ajout nouveau chapître “Astuces (howto)”
- 29/9/20 Mise à jour du chapitre “Installations existantes” pour ajouter IP2I
- 28/9/20 Ajout d’un nouveau chapitre expliquant la gestion des notifications
- 8/9/20 Ajout d’un exemple dans le howto “Ajouter une table dans la BD” (ajout de la nouvelle table “**projets**”)
- 4/9/20 Mise à jour du chapitre “Etiquettes” suite à la mise en place de la nouvelle étiqueteuse Dymo MobileLabeler
- 24/7/20 Mise à jour du chapitre “Installations existantes” pour IAS
- 23/7/20 Ajout de CETTE page-ci
- 23/7/20 Modification/Ajout des chapitres suivants :
    - Configuration des autorisations
    - Configuration des logos
    - Configuration des étiquettes
    - Configuration niveau DEV only

TABLE DES MATIÈRES

1. LICENCE

Auteurs : Etienne Pallier (epallier@irap.omp.eu), Elodie Bourrec (ebourrec@irap.omp.eu)

Le logiciel **LabInvent** (InventIrap pour l'IRAP) est sous licence libre copyleft **AGPL** (**GNU Affero General Public License**) v3.0 dont le détail est donné sur la page web https://spdx.org/licenses/AGPL-3.0.html#licenseText, mais aussi dans le fichier texte "LICENSE AGPL" à la racine du projet.

(voir aussi https://choosealicense.com/licenses/agpl-3.0, https://www.diatem.net/les-licences-open-source, et https://www.gnu.org/licenses/why-affero-gpl.fr.html, ainsi que https://fr.wikipedia.org/wiki/GNU_Affero_General_Public_License)

Ce logiciel est développé depuis 2012, à l'origine pour les besoins du laboratoire IRAP de Toulouse, sous le nom dédié "Inventirap". Depuis, il a été diffusé dans d'autres laboratoires, avec l'appellation plus générique de "LabInvent". Il est construit sur un framework Php orienté objets nommé "CakePhp", dans sa version 3.x (http://cakephp.org) qui n'est pas inclut mais récupéré automatiquement au moment de l'installation. Le framework CakePHP est sous licence MIT (licence sans copyleft). Il fonctionne avec Php 7 (mais reste encore compatible avec Php 5.6+)

Bien qu'il soit développé avec une attention particulière portée sur la qualité, ce logiciel est mis à disposition "en l'état" ("as is"), sans garantie aucune.

Toute modification n'altérant pas la finalité principale du logiciel qui est d'inventorier les matériels, est autorisée. Elle doit toutefois être partagée et ré-injectée dans le logiciel, afin que toute la communauté des utilisateurs puisse en profiter, et afin que le logiciel LabInvent reste une entité unique et bien définie.

2. Vérification des pré-requis

Avant de pouvoir utiliser correctement le logiciel, vérifiez que tout est bien en place.
Pour cela, on peut se référer à la documentation d'installation, surtout le chapitre qui traite des pré-requis.

3. Contribuer - A quel niveau intervenir selon les besoins (où faire quoi ?)

\oufairekoi

Cette section répond au besoin de **savoir à quel niveau on doit intervenir selon la modification qu'on souhaite réaliser**.

Elle est destinée aux **personnes qui souhaitent contribuer** au logiciel mais ne savent pas où agir.

Il y a 6 niveaux principaux d'intervention selon ce qu'on souhaite faire :

- **DOC** : on souhaite améliorer l'information sur le logiciel => il suffit pour cela de compléter cette présente documentation
- **INSTALL** : on souhaite améliorer la phase d'installation du logiciel => il faut agir sur le script **install/install.sh** (on pourrait aussi traduire ce script en php pour une compatibilité multi-plateformes) => voir le chapitre "installation du logiciel"
- **TESTS** : on souhaite renforcer la qualité du logiciel => il faut pour cela renforcer la suite de tests déjà existante (voir le chapitre sur les tests)
- **PERF** : on souhaite accélérer l'application => on peut paralléliser certaines tâches telles que l'envoi d'emails, la génération des pdfs, l'exportation <des listes de matériels, les traitements automatiques tels que la mise en cache du LDAP ou le processus de relance par mail pour les suivis…
- **AUTORISATIONS** : on souhaite modifier les droits des différents profils associés aux utilisateurs (qui a le droit de faire quoi) => voir le chapitre sur le workflow
- **CODE** : on souhaite modifier l'apparence ou le comportement du logiciel => il faut pour cela agir sur le code source

Nous allons maintenant détailler le dernier niveau d'intervention ci-dessus qui concerne la **modification du code source**. La modification du code source **se subdivise elle-même en 3 niveaux** selon ce qu'on désire modifier. Ces 3 niveaux correspondent à ceux du **pattern "MVC" (Modèle/Vue/Controleur)** sur lequel le framework cakephp est basé. En modifiant le logiciel, on agira :

- soit sur l'aspect (présentation) : qui peut Voir quoi => niveau (V)ue
- soit sur le comportement : qui peut Faire quoi => niveau (C)ontrolêur
- soit sur les entités et leurs relations : de quoi parle le logiciel => niveau (M)odèle

3.1. Modification de l'ASPECT du logiciel => Niveau "Vue" du MVC

En agissant à ce niveau, on modifiera **l'apparence** du logiciel, sa **présentation**.

C'est le niveau "le plus intéressant" car toute modification à ce niveau est immédiatement visible, c'est donc plus gratifiant que les 2 autres niveaux suivants.

A ce niveau, on définit à la fois :

- la structure, la présentation, et le contenu des différentes pages web de l'application, en gros "où on met quoi et comment",
- et aussi "qui peut Voir quoi" (le **qpVq**)

Exemples :

- tout le monde peut voir qui a créé ou modifié une fiche matériel => actuellement ça n'est pas le cas, ça n'est visible qu'à un profil "superadmin", mais on pourrait changer ça à ce niveau
- faire en sorte que l'application soit visible sur un petit écran (smartphone) => ceci a été fait début 2020
- paginer la liste des matériels
- afficher le QrCode en haut à droite de toutes les fiches matériel
- quand on visualise une fiche matériel, on doit aussi voir la liste des ses suivis et des ses emprunts
- … etc

Pour intervenir à ce niveau, les compétences techniques suivantes sont nécessaires :

- **Php** (niveau assez basique) : génération dynamique du code html, et insertion des données dans les pages, en tenant compte du profil de la personne connectée, de la configuration du logiciel, de l'état des données présentes
- **Html** : structure et organisation des pages
- **Css** : aspect des pages
- **Javascript** : comportement dynamique des pages

3.2. Modification du COMPORTEMENT du logiciel (règles métier) => Niveau “Contrôleur” du MVC

En agissant à ce niveau, on définit **ce qui est possible et par qui**, en gros “qui peut Faire quoi” (le **qpFq**).

En simplifiant à outrance, on pourrait dire qu’on définit ce qui est “read only” et pour qui.

Dans le détail, on définit à ce niveau à la fois :

- les **ACTIONS** possibles sur les divers “objets” (tables de la BD) de l’application,
- et **QUI** a le droit de les exécuter.

L’ensemble des actions et des droits qui sont associés constituent ce qu’on appelle “les **règles métier**” du domaine étudié, en l’occurence ici un “inventaire des matériels”.

Il existe **4 actions fondamentales** sur les différents “objets” (tables de la BD) de l’application, les actions nommées **CRUD pour Create, Read, Update, et Delete**. A ces actions fondamentales, on peut en ajouter d’autres plus spécifiques qui correspondent au domaine métier de l’application. Par exemple, pour gérer le statut d’un matériel, on a besoin de modifier son statut en le validant, ou encore en l’archivant, ce qui correspond à 2 nouvelles actions “valider” et “archiver”, qui sont vraiment spécifiques à un inventaire matériel.

Exemples d’actions et droits associés :

- tout le monde peut **créer** une nouvelle fiche matériel
- tout le monde peut **modifier** tous les matériels => actuellement, ce n’est pas le cas, seul l’utilisateur d’un matériel peut modifier sa fiche, ou alors il faut avoir un profil supérieur de type “admin” ou “superadmin”
- le profil “admin” peut **modifier** un matériel validé
- on ne peut pas **supprimer** la fiche d’un matériel qui est validé
- ...

Tous ces exemples représentent des **règles métier**.

La couche “contrôleur” fait l’intermédiaire entre les 2 autres couches (modèle et vue). Elle a pour rôle de répondre aux requêtes de l’utilisateur en récupérant les données nécessaires via la couche “Modèle”, et en les organisant avant de les fournir à la couche “Vue” pour qu’elles soient présentées à l’utilisateur.

A ce niveau les compétences techniques requises sont :

- **Php** (plus haut niveau, orienté objet)

3.3. Modification des données (entités) et de leurs relations => Niveau “Modèle” du MVC

A ce niveau, on définit les **entités** (objets, tables de la BD) du domaine étudié (ici un “inventaire”) et leurs **relations**. En quelque sorte on “modélise” le domaine étudié. Par exemple, un inventaire est constitué des **entités** “matériel”, “utilisateur”, “emprunts”, “suivis”, “qrcode”, etc. Dans un tel “modèle”, on doit créer les **relations** entre les entités qui reflètent le fait qu’un matériel “appartient” à un “utilisateur”, qu’il a été “créé” par tel autre, que ce matériel est de telle catégorie, qu’un utilisateur a tel profil, que ce matériel a été emprunté par tel ou tel utilisateur, qu’il est suivi pour une intervention technique, etc…

C’est aussi à ce niveau qu’on définit les “**règles de validation**” des diverses entités du domaine, c’est à dire tout ce qui doit être vérifié pour dire qu’une entité a été correctement saisie par l’utilisateur. Seule une entité validée peut être sauvegardée dans la BD.

Exemples :

- ajouter un attribut à l’entité “matériel” : couleur, dimension, poids, adresse Mac ou IP, etc.
- ajouter une nouvelle “entité”, par exemple la notion de “type de matériel”
- ajouter la gestion des entités abstraites telles que les logiciels, les licences…
- ajouter une relation de composition entre matériels permettant de dire qu’un matériel est composé de tel et tel autre, qui sont ses composants

A ce niveau les compétences techniques requises sont :

- **Php** (plus haut niveau, orienté objet)
- **Modèle relationnel**
- **Requêtes Sql**

C’est sans doute cette couche qui est la plus sensible et dans laquelle toute modification doit être faite avec une extrême prudence.

4. Les différents MODES du logiciel (panique, installation, debug, avec ou sans ldap)

A ce sujet, voir la documentation technique générale

5. Configuration technique du logiciel (Adaptation)

Nous décrivons la configuration des **paramètres techniques** du logiciel, tels que la configuration de la base de données, ou des autorisations par profil utilisateur...
Pour la configuration des **paramètres métier**, via des pages web dédiées, voir la documentation technique générale.

5.1. Configuration technique via le fichier de configuration générale config/app.php

Cette configuration permet de modifier les paramètres techniques de base (BDD, emails, mode debug général, …).
Elle se fait via la modification du fichier de configuration générale "**config/app.php**".

Normalement, vous ne devriez pas avoir besoin de faire ça, sauf pour passer en **mode DEBUG général** (ou pour supprimer ce mode), auquel cas vous pouvez aller à la section dédiée de ce document.

C'est mieux de rester en mode DEBUG au début de l'installation du logiciel, le temps de voir si tout marche bien…

On pourrait ajouter d'autres paramètres dans ce fichier, selon le besoin, mais en général on préférera passer par la **page web de configuration générale**. C'est ce qui est présenté dans la section suivante.

5.2. Configuration des Autorisations selon les profils utilisateurs (ACLs, authorizations)

(EP updated 03/12/2020)
\acl \authorization \profil

Des autorisations différentes sont accordées à chaque profil utilisateur.

Chaque utilisateur du logiciel a par défaut le profil le plus bas, c'est à dire "Utilisateur".

Les utilisateurs privilégiés ont des profils privilégiés tels que (dans l'ordre croissant des pouvoirs) "Responsable", "Administration", et "Super Administrateur" (qui est un profil système ayant quasiment tous les droits).

Les autorisations sont accordées à 2 niveaux :

- 1) d'abord au niveau de **l'ACTION** demandée, c'est assez basique : la question est **juste de savoir si l'action demandée est autorisée ou pas** pour ce profil utilisateur ? (si oui, l'accès est accordé, sinon, il est refusé)
- 2) puis au niveau de la **VUE associée à cette action**, si elle existe (car certaines actions n'ont pas de vue associée, par exemple l'action "supprimer un matériel" se contente de supprimer le matériel, pas besoin d'afficher quoi que ce soit ; par contre, l'action "modifier un matériel" a la vue "materiels/edit" associée qui affiche tous les attributs du matériel que l'on peut alors modifier) ; au niveau de la vue il faut **définir des autorisations presque pour chaque attribut :** pour le profil utilisateur en cours, certains attributs (champs) doivent-ils être cachés, certains sont-ils en lecture seule, certains ont-ils une valeur par défaut et/ou une contrainte associée (ex: date livraison doit être supérieure à date achat…)

5.2.1. Autorisations pour le niveau 1 (ACTION)

Pour le niveau 1, les autorisations accordées à chaque profil sont **affichées sur la page web des "Autorisations"** (menu Outils, "Voir les Autorisations des profils utilisateurs" ou directement /pages/acls). **La procédure à suivre pour modifier ces autorisations est directement indiquée sur cette page**. Temporairement (pour des besoins de débogage) on peut accorder quasiment "tous les droits" au profil SuperAdmin, en surpassant les droits actuellement en vigueur pour ce profil. Pour cela, il suffit de cliquer sur le lien "Activer le mode 'Superadmin a tous les droits'"dans la page Outils (ce lien devient ensuite "Désactiver le mode …").
Anciens documents (pas à jour) décrivant ces autorisations pour chaque entité (Materiels, Suivis, Emprunts, Utilisateurs, …) :

⇒ Tableau Excel (GDoc) des autorisations telles que définies à l'origine (pas à jour)
⇒ Ancien document décrivant le système des autorisations

5.2.2. Autorisations pour le niveau 2 (VUE)

(EP 7/12/20)

Pour le niveau 2 (vue), les autorisations sont définies **pour chaque attribut** (ou champ) d'une entité, et **principalement pour les vues correspondant aux actions add (ajout) et edit** (modification), ces 2 actions étant souvent fusionnées en une seule "**add_edit**" (ajout ou modification) car très semblables. Donc nous nous concentrerons principalement sur la **vue "add_edit**". Un exemple est donné ci-dessous **pour l'entité la plus importante,** à savoir l'entité **"Materiel"** (prochainement, on pourrait se livrer au même exercice pour d'autres entités secondaires, mais néanmoins importantes, telles que Utilisateur, Suivi, Emprunt, Domaine, etc.).

A ce niveau (vue, et essentiellement la vue "add_edit"), il est **beaucoup plus difficile (voire impossible) de centraliser les autorisations**, comme on a pu le faire au niveau 1 (action). En effet, et comme le tableau ci-dessous le montre bien, on autorise, POUR CHAQUE ATTRIBUT de l'entité, et pour différents types de choses, telles que :

- (1) cet attribut est-il obligatoire ?
- (2) a-t-il un domaine de définition précis, c'est à dire un ensemble de valeurs possibles que l'on peut lister ?
- (3) a-t-il une valeur par défaut ?
- (4) est-il en lecture seule ?
- (5) et enfin, existe-t-il sur lui une (ou des) contrainte d'intégrité ou de cohérence ou une règle métier à respecter (ex: la date de livraison doit être supérieure à la date d'achat) ?

Le dernier type (5) est plus ou moins un intrus dans cette liste car c'est le seul qui soit plutôt indépendant du profil utilisateur. En effet, il représente une contrainte qui s'applique en général à plusieurs attributs à la fois, et qui permet de maintenir la cohérence de ces attributs. Je l'ai laissé ici car il permet d'être exhaustif sur la description de toutes les caractéristiques des attributs. Mais en ce qui concerne la notion d'autorisation liée à un profil utilisateur, nous allons la laisser de côté*.

Chacun des 4 autres types d'autorisation peut avoir une **valeur différente selon le profil de l'utilisateur**, c'est pourquoi on les considère bien comme des "types d'autorisation", bien que ces contraintes existeraient quand même dans un système SANS profil utilisateur.

Chacun de ces types d'autorisation (1 à 5) donne lieu à une colonne (1 à 5) dans l'exemple ci-dessous. Et on peut voir sur la ligne "Emplacement de définition" que ces types d'autorisation ne sont pas définis au même emplacement (ou niveau), ce qui complique leur mise à jour (maintenance)... En effet, les types 1 et 5 sont définis au niveau de la classe Table (et dans 2 fonctions différentes !), alors

que les types 2 à 4 peuvent être définis directement dans la vue, mais il est recommandé de les définir dans le contrôleur. On a donc en gros 2 emplacements de définition : 2 types sur 5 dans la classe Table de l'entité (mais dans 2 fonctions différentes) et 3 types sur 5 dans le Contrôleur de l'entité. On peut donc centraliser 60% (3/5e) des autorisations dans le Contrôleur (et même 80%, 4/5e, si on considère que le LOT2 (voire même le LOT1) sera aussi défini dans le Contrôleur, voir ci-dessous ; on atteint même 100% si on laisse de côté le type 5 comme suggéré plus haut*), et le reste (40%) dans la classe Table, ce qui n'est déjà pas si mal...

Exemple : Autorisations pour les Vues "ajout" et "édition" (modif) de **l'entité Materiel** (ces 2 vues sont très semblables et d'ailleurs elles sont fusionnées en une seule dans le code source)

Légende :

- EDITEUR = l'utilisateur qui édite la fiche
- La "section administrative" n'est éditable que par un profil ADMIN (Gestionnaire)
- LOT1 = données obligatoires pour passer commande (restent toujours obligatoires ensuite), quand on clique sur le bouton "Enregistrer & Commander"
- LOT2 = données obligatoires pour valider la fiche : matériel livré et payé (restent toujours obligatoires ensuite), quand on clique sur le bouton "Valider (livré et payé)" ; ces infos ne peuvent être saisies/modifiées que par un profil ADMIN
- A partir du statut VALIDATED :
    - les champs du LOT1+LOT2 ne sont plus modifiables, sauf "utilisateur"
    - les docs attachés ne peuvent plus être supprimés (mais on peut en ajouter)

Champ (attribut)	**1.** **Obligatoire**	**2.** **Options (valeurs possibles)**	**3.** **Défaut** *(add only)*	**4.** **Read-only (R)** *(edit only)* **ou Caché (C)** *(add ou edit)*	**5.** **Contrainte**
Emplacement de définition :	*dans la classe **Table** (méthode **validationDefault**() avec allowEmpty(false)) (mais on pourrait aussi tout rendre facultatif et définir ça dans le Contrôleur au moment du test du*	*dans le **Contrôleur** qui passe à la vue (via **set**()) une variable du même nom mais au pluriel*	*dans le **Contrôleur** qui initialise le champ de **l'entité** avec la valeur par défaut*	*dans la **Vue** (mais ça serait mieux dans le **Contrôleur**, à travers des variables telles $is_readonly_field et $is_hidden_field)*	*dans la classe **Table** (méthode **buildRules**() avec une règle custom, custom rule)*

	POST) *(ok pour **LOT1**, c'est à dire les champs obligatoires par défaut ; par contre pour **LOT2** c'est plutôt défini dans le **Contrôleur,** voire même aussi LOT1...)*				
LOT1 :					
designation	X (**LOT1**)				
permanent	X (LOT1)	oui/non	oui		
domaine (sur_categorie)	X (LOT1)	**surCategories**			
categorie	X (LOT1)	**categories**			
sous_categorie		**sousCategories**			
n° inventaire (numero_laboratoire) IRAP-2020-0087)	X (LOT1)			**C** **Calculé auto =** labo_short_name + Année de date_achat (ou année en cours) + numéro séquentiel	
description (important pour communiquer avec le Gestionnaire)	X (LOT1)				
prix_ht	X (LOT1)				> 0

acheteur (nom_responsable)	X (LOT1)	**utilisateurs** (from LDAP)	EDITEUR	R (si profil USER)	
email acheteur (email_responsable)	X (LOT1)			R (calculé, récupéré dans les infos de l'acheteur)	
resp. crédit (si autre que acheteur)	X (LOT1)	(champ texte libre)			
utilisateur final (nom_user)	X (LOT1)	(champ texte libre)	EDITEUR		
gestionnaire (de ref.)	X (LOT1)	**gestionnaires**	EDITEUR (si gestionnaire) sinon : "Je ne sais pas"		
organisme	X (LOT1)	**organismes** (CNRS, UPS, …)			
fournisseur	X (LOT1)	**fournisseurs** (+ ajout auto si nouveau)			
devis joint (doc joint type DEVIS)	X (LOT1+) (désactivable par configuration)				
LOT2 :					
date_acquisition (= date du BC)	X (LOT2)		aujourd'hui (si c'est un Gestionnaire qui édite)		pas future (et pas trop ancienne)
facture jointe (doc joint type FACTURE)	X (**LOT2**) si prix > 10K€				si prix > 10K€

[numero_serie]	X (LOT2) si prix > 10K€				si prix > 10K€
date livraison (date_reception)	X (LOT2)				>= date_achat (mais pas trop loin dans le futur)
etiquette (si titreuse configurée) (= etiquette imprimée?)	X (LOT2)	oui/non	non		
lieu_stockage (site)	X (LOT2)	**sites**			
lieu_detail (detail)	X (LOT2)				
groupe_thematique		**groupes_thematique**			
groupe_metier		**groupes_metier**			
projet		**projets**			
SECTION ADMINISTRATIVE & BUDGÉTAIRE : (profil ADMIN only)					
Ligne budgétaire / Entité dépensière (eotp) *LOT1 (acheteur) => “Sur quel(s) budget(s) ?” (ligne budgétaire)* *LOT2 (gestionnaire) => “Entité(s) dépensière(s)”*	X (LOT1)			R (si pas profil ADMIN)	
numero_commande (n° BC)	X (LOT2)			R (si pas profil ADMIN)	

tutelle (= service fait) (numero_inventaire_organisme)	X (LOT2)			R (si pas profil ADMIN)	
AUTRES ATTRIBUTS (Hors Lot1 et Lot2) :					
status	X	CREATED, VALIDATED, TBA, ARCHIVED	CREATED	C (calculé auto pour les changements d'état)	CREATED < VALIDATED < TBA < ARCHIVED
à commander (**tobeordered**)	X	oui/non	non	C	
duree_garantie					
unite_duree_garantie		mois, ans	ans		
date_fin_garantie				R (calculé = date achat + durée garantie)	> date livraison
hors_service	X	oui/non	non		

5.3. Configuration des Notifications (log et email)

(EP updated 28/9/20)

Le système de gestion des notification qui gère les logs et l'envoi d'emails a été entièrement remanié fin septembre 2020.

Désormais, toute action (création, modification, suppression...) faite sur toute entité (Materiel, Document, Suivi, Emprunt, ...) peut faire l'objet d'une notification, celle-ci étant faite soit par mail, soit par log, soit par les 2 moyens (mail et log).

Par défaut, seules quelques actions de quelques entités importantes (matériels et documents) envoient des notifications.
Cette liste des actions "notifiantes" est affichée sur une page web (/pages/notifications), accessible depuis la page "Outils".

Cette liste est entièrement configurable, et indépendamment pour chaque laboratoire (c'est à dire, pour chaque instance du logiciel LabInvent).
La configuration se fait via le code source (comme pour les règles d'accès, pas encore via la BD, ca viendra peut-être un jour...), et est expliquée directement sur la page des notifications (/pages/notifications).

Les notifications par log ne sont pas désactivables de manière globale.
Seuls les notifications par mail peuvent être entièrement désactivées.

Par défaut, les notifications par email sont envoyées :

- à l'utilisateur destinataire du matériel
- au gestionnaire de référence du matériel
- aux responsables (thématique ou métier) du matériel
- et aussi à une liste de mails prédéfinie et configurable via la page web de configuration générale
- (TODO) peut-être plus tard, aux responsables (scientifique et chef projet) du projet auquel le matériel est associé

Bien sûr, l'auteur de l'action n'est pas notifié (il est déjà au courant...)

5.3.1. Activation/Désactivation des notifications par mail

Pour activer ou désactiver l'envoi d'email, aller sur la page de configuration générale du logiciel, depuis le menu "Outils" :
En tant que SuperAdministrateur, aller dans "Outils" -> "Configuration générale de l'application", cliquez sur "Editer la configuration".
Vous avez alors 2 options :

- "Activer l'envoi des mails général" : active l'envoi d'emails à la liste générale, c'est à dire aux responsables de groupes (thématique, métier, …), aux responsables d'un matériel, au personnel administratif, etc.
- "Activer l'envoi des mails pour la liste spécifique ci-dessous" : active l'envoi d'emails à cette liste spécifique

Si ces 2 options sont cochées, les emails sont envoyés aux 2 listes (liste générale et liste spécifique).

5.3.2. Configuration des actions notifiantes, pour chaque entité

Cette configuration se fait (comme pour les configuration des Autorisations) via le code source du projet, indépendamment pour chaque entité pour laquelle vous désirez des notifications (sur certaines actions).

Pour modifier les notifications concernant une entité spécifique (matériels, documents, suivis, emprunts, etc.), aller dans la classe contrôleur de cette entité.

Par exemple, pour changer les notifications liées aux matériels (entité Materiels), aller dans le fichier /src/Controller/MaterielsController.php :

- pour modifier les règles (LOCALES de votre laboratoire seulement), aller dans la fonction setAuthorizations_XXX() => elles ne s'appliqueront que au laboratoire XXX (le vôtre bien sûr)
- pour modifier les règles (GÉNÉRALES, valables pour TOUS les laboratoires), aller dans la fonction setAuthorizations() => elles s'appliqueront à TOUS les laboratoires

Dans la fonction setAuthorizations choisie, modifier les arguments de la fonction setNotificationAllowedOnActions()

Veuillez ensuite partager ces modifications en les intégrant au code source général du logiciel (ainsi que toute autre modification effectuée) via la commande :

```
$ ./PUSH_MODIFS
```

5.3.3. Mécanisme interne d’envoi des emails

L'envoi de mail est réalisé grâce à une adresse qu'il faut créer pour LabInvent. Le protocole d'envoi est à définir dans config/app.php selon le serveur choisi. Lors de l'envoi d'un mail il faut utiliser le transport 'dev' en local.

Pour changer de mail et de transport, il faut aller dans la section Email/Transport de config/app.php, et vérifier/adapter a votre convenance le bloc suivant :

```
'EmailTransport' => [
        'default' => [
            'className' => 'Mail',
            // The following keys are used in SMTP transports
            'host' => 'localhost',
            'port' => 25,
            'timeout' => 30,
            'username' => 'user',
            'password' => 'secret',
            'client' => null,
            'tls' => null,
            'url' => env('EMAIL_TRANSPORT_DEFAULT_URL', null),
        ],
…
```

5.4. Ajouter des paramètres de Configuration

5.4.1. Utilisation des paramètres de configuration dans le code (dynamiquement)

- **dans un contrôleur** (ex: src/Controller/Pages/PagesController.php) :
  - // La variable $this->**confLabinvent** est **définie** dans la classe mère **AppController** (pour que tous les contrôleurs en héritent), dans la fonction **initialize**(), comme ceci :
    $this->confLabinvent = **TableRegistry::get('Configurations')->find()->first();**
  - // Elle est donc utilisable dans n'importe quel contrôleur
    // ex: pour savoir si on est en mode INSTALL ou pas :
    $configuration = $this->**confLabinvent**
    if ($configuration->mode_install) …
  - // Elle est aussi **passée à TOUTES les vues** (templates) via la fonction **beforeRender**() de AppController, sous la forme d'une variable nommée $**configuration** :
    $configuration = $this->confLabinvent;
    $this->**set**(**'configuration'**, $configuration);

- **dans un template** (ex: src/Template/Pages/tools_sm.ctp), la variable $configuration (passée à toutes les vues par AppController comme expliqué ci-dessus) est disponible :
  // ex: pour savoir si la variable de configuration "metrologie" est à True ou False :
  if ($configuration->metrologie) ...

5.4.2. Comment ajouter un nouveau paramètre de configuration

(TODO: utiliser le principe des "Migrations" cakephp, ça éviterait d'avoir à toucher la BD à la mano)

Si vous désirez ajouter un nouveau paramètre de configuration, voici la procédure à suivre.

Prenons l'exemple suivant : on veut ajouter une option "format d'étiquette" pour permettre à un laboratoire de choisir son format d'étiquette à imprimer. Pour cela, on va ajouter un nouveau champ nommé "label_format_num".

a - (Optionnel) Ajouter une entrée pour valider le nouveau champ “label_format_num” de la table configurations

Editer le fichier src/Model/Table/ConfigurationsTable.php
Dans la fonction validationDefault(), ajouter une ligne :

```
$validator->allowEmpty(label_format_num);
```

b - Ajouter une colonne “label_format_num” dans la table configurations

On peut utiliser phpmyadmin pour ça, ou encore la commande “mysql -u” en mode terminal.
Exécuter cette requete SQL :

```
ALTER TABLE `configurations` ADD `label_format_num` INT(3) UNSIGNED NOT NULL DEFAULT '1' COMMENT 'numero de format etiquette'
AFTER `hasPrinter`;
```

c - Ajouter cette requete SQL dans un fichier de mise à jour de la BD

Ceci, afin que les autres utilisateurs du logiciel puissent hériter (automatiquement) de cette nouvelle fonctionnalité lors de la prochaine exécution du script UPDATE.
Aller dans database/update/

Faire une copie du DERNIER script BASH présent, par exemple db-update-2019-01-09.sh, et le nommer à la date du jour : db-update-2019-03-14.sh
Pas besoin de modifier ce fichier, il faut juste qu’il soit présent.

Ce fichier exécutera sur la BD la requete SQL du même nom qui se trouve dans le sous-dossier script_sql/.
Donc, dans ce sous-dossier, de la même façon, faire une copie du DERNIER script SQL, par exemple db-update-2019-01-09.sql et le nommer à la date du jour : db-update-2019-03-14.sql
Attention, les 2 scripts (bash et sql) doivent être à la MEME DATE.
Dans votre script SQL db-update-2019-03-14.sql, ajouter le code SQL de l’étape précédente.
Ce script doit donc avoir le contenu suivant :

```
use database;

ALTER TABLE `configurations` ADD `label_format_num` INT(3) UNSIGNED NOT NULL DEFAULT '1' COMMENT 'numero de format etiquette'
AFTER `hasPrinter`;
```

d - Ajouter le champ “label_format_num” dans la vue d’édition de la table “configurations”

Editer le fichier src/Template/Configurations/**edit**.ctp
Ajouter ces lignes sous la ligne qui concerne le champ “hasPrinter” :

```
echo $this->Form->control('label_format_num', [
    'options' => [
        '1' => 1,
        '2' => 2,
        '3' => 3,
        '4' => 4,
        '5' => 5,
        '6' => 6,
    ],
    'label' => 'Numéro Format Etiquette'
]);
```

e - Ajouter le champ “label_format_num” dans la vue détaillée de la table “configurations”

Editer le fichier src/Template/Configurations/**view**.ctp

Sous la ligne qui concerne l’imprimante (“Imprimante disponible”), ajouter la ligne suivante :

```
$displayElement(__('Numéro format étiquette'), h($configurationObj->label_format_num));
```

f - Utiliser cette nouvelle option dans le code qui en dépend

En l’occurence, c’est le contrôleur des matériels qui utilise cette option.
Editer le fichier src/Controller/MaterielsController.php
Aller dans la fonction printLabel()
Ajouter cette ligne pour récupérer la valeur actuelle de l’option (dans la table configurations) :

```
$label_format_num = $this->confLabinvent->label_format_num;
```

g - Documenter cette nouvelle option dans CE document

Si si, c'est important ça, vous êtes pas tout seul au monde, m'enfin, allez, au boulot !!!
https://tinyurl.com/labinvent

Puis, faire une copie PDF "Labinvent Documentation.pdf" de ce document (Google Doc) et la mettre dans le dossier doc/

h - Commiter tous ces changements sur le dépôt GIT central du projet

$ git pull
$ git add .
$ git commit -m "Ajout d'une nouvelle option pour choisir le format d'étiquette"
$ git push

Voilà, les laboratoires qui voudront récupérer cette nouvelle option auront seulement à exécuter le script "./UPDATE" (à la racine du projet).
Elle est pas belle la vie ?

6. Adaptation des étiquettes au besoin du laboratoire

(updated 25/11/20 - EP)
\etiquet \titreuse \print \imprim

Pour les considérations générales concernant l'utilisation d'une étiqueteuse, voir la documentation technique.

Si vous voulez un format d'étiquette différent de celui proposé par défaut, vous pouvez créer le vôtre en suivant toutes ces étapes.
Cette section a besoin d'être un peu actualisée. Elle n'est pas complètement juste mais reste néanmoins utilisable.

a - Créer votre étiquette idéale avec le logiciel propriétaire DCD (ou DLS)

Ouvrir le logiciel DCD (ou DLS).
Choisir le format d'étiquette voulu dans le panneau de gauche (**19mm étant celui utilisé par défaut pour LabInvent**).
Créer votre étiquette comme souhaitée dans le panneau de droite.
Enregistrer cette étiquette dans un dossier quelconque avec "Fichier/Enregistrer sous…"
Ce fichier porte l'extension ".label" et est au format XML.
Voici un extrait du début d'un fichier de ce type :

```
<?xml version="1.0" encoding="utf-8"?>
<ContinuousLabel Version="8.0" Units="twips">
  <PaperOrientation>Landscape</PaperOrientation>
  <Id>Tape19mm</Id>
  <PaperName>19mm</PaperName>
  <LengthMode>Auto</LengthMode>
  <LabelLength>0</LabelLength>
  <RootCell>
        <Length>0</Length>
        <LengthMode>Auto</LengthMode>
```

b - Créer votre fonction étiquette dans le code source du logiciel LabInvent (copier le contenu XML de votre fichier étiquette)

Oui, je sais, c'est pas du tout l'idéal, mais je sais pas comment faire mieux pour l'instant.
Aller dans le contrôleur des matériels (src/Controller/**MaterielsController.php)**
Faire une copie de la fonction **etiquette_format1**() qui est la définition du format d'étiquette pour le **laboratoire IRAP**. Cette fonction sera automatiquement appelée par la fonction printLabel().
Nommer cette nouvelle fonction etiquette_format2() (ou etiquette_format3... si etiquette_format2 existe déjà).
Modifier les commentaires juste au-dessus de cette fonction pour décrire le format (ruban 12mm, 1, 2, ou 3 lignes, avec ou sans logo...)

Dans VOTRE nouvelle fonction **etiquette_format2**(), Il faut copier TOUT le contenu XML de votre fichier étiquette, à la place du code existant.
Attention toutefois, cette fonction utilise 4 paramètres qui permettent de faire varier le contenu de l'étiquette :

- $numeroLab
- $organisme
- $dateAcquisition
- $numeroInventaireOrganisme

Ce sont ces paramètres qui sont imprimés par défaut sur les étiquettes.
Si vous voulez imprimer d'autres paramètres, il faudra les changer, et vous assurer qu'ils sont bien passés à cette fonction lors de son appel dans la fonction printLabel()

Sur l'image ci-dessous de l'étiquette IRAP, on remarque qu'elle est organisée en 2 colonnes :

- une première colonne (à gauche sur l'étiquette) contenant le texte (formé des 4 paramètres mentionnés ci-dessus).
- une deuxième colonne (à droite sur l'étiquette) contenant une image, le logo IRAP

Voir ci-dessous le contenu XML correspondant à cette étiquette. On retrouve ce contenu dans la fonction **etiquette_format1**().
Dans ce code XML, les colonnes (vues sur l'étiquette ci-dessus) sont appelées cellules (<**Cell**>).
Il y a donc 2 sections <Cell> :

- une première (de type <**TextObject**>) qui décrit les lignes de texte
- une deuxième (de type <**ImageObject**>) qui décrit l'image (logo IRAP)

Le texte à imprimer est défini au début de la fonction **etiquette_format1() avec $text_line1 et $text_line2**
L'image à imprimer est définie avec $**img_logo**.
Ce sont ces variables que vous devrez adapter pour vos besoins.

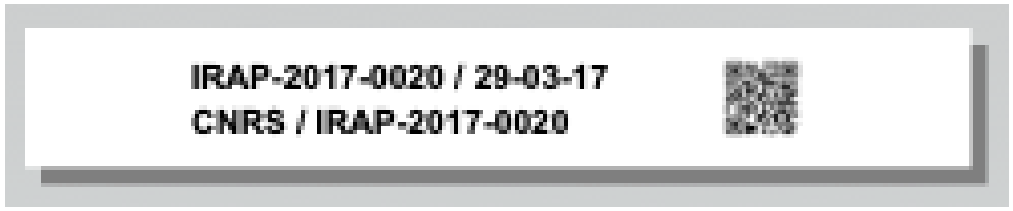
```
<Subcells>

    <Cell>
        <TextObject>
        …
            <StyledText>
                <Element>
                    <String>'
                        .$text_line1 . "\n"
                        .$text_line2 .
                    '</String>
                    <Attributes>
                        <Font Family="Arial" Size="24" Bold="True" Italic="False" Underline="False" Strikeout="False"/>
                        <ForeColor Alpha="255" Red="0" Green="0" Blue="0"/>
                    </Attributes>
                </Element>
            </StyledText>
        </TextObject>
        ...
    </Cell>

    <Cell>
        <ImageObject>
            ...
            <Image>'
                .$img_logo.
            '</Image>
            ...
        </ImageObject>
        …
    </Cell>

</Subcells>
```

Une autre fonction “etiquette” nommée etiquette_format_avec_QRCODE() fournit un exemple avec la même étiquette IRAP ci-dessus mais qui remplace le logo IRAP par le QRCode du matériel. Voici l’étiquette produite alors :



c - Sélectionner votre étiquette dans la partie configuration du logiciel

Une fois votre fonction étiquette **etiquette_format2()** bien définie, vous devez la sélectionner en allant dans le menu configuration du logiciel :

Aller dans le menu “Outils” puis “Configuration générale de l’application”.
Cliquer sur “Editer la configuration”.
Aller dans la section “Divers”.
S’assurer que l’option “Imprimante disponible” est bien cochée.
Dans l’option “**Numéro format étiquette**”, sélectionner le numéro “2” correspondant à votre fonction **etiquette_format2()** que vous avez créée (ou bien, sélectionner “3” pour la fonction etiquette_format3()**,** “4” pour la fonction etiquette_format4(), etc.)
Cliquer sur le bouton vert “Valider” tout en bas.

d - Tester

Aller sur la vue détaillée d’un matériel. Attention, on ne peut imprimer l’étiquette que d’un matériel **VALIDÉ**.
Cliquer sur le bouton “Impr. etiquette”. Voilà !

7. Gestion des utilisateurs (Connexion, profils associés, LDAP)

(updated 8/2/21 - EP)

\user \ldap \auth \acl \autorisation \connexion \profil \role

7.1. Le cache LDAP - Explication du fonctionnement (mise en BD de la liste des utilisateurs de l’annuaire LDAP)

Niveau : D
(updated 4/2/19 - EP)

Où se trouve le code qui affiche la liste des utilisateurs dans les vues ajout/édition de matériel ?
Il suffit de suivre cette logique :

- Ajout de materiels = src/Template/Materiels/add.ctp
- Edition de materiels = src/Template/Materiels/edit.ctp

Dans la vue "add.ctp" (et “edit.ctp”) tu trouves le champ "Nom de l\'utilisateur"
Il contient une variable 'options' => $utilisateurs
C'est donc cette variable $utilisateurs qui est utilisée.

Là ca se complique un peu :
Cette variable n'est pas définie dans la vue, mais dans le controlleur de matériels qui l'a créée et passée à la vue :
On cherche donc "$utilisateurs" dans src/Controller/MaterielsController.php, on le trouve à la ligne 811 (elle est ensuite passée à la vue à la ligne 830 avec set(..., 'utilisateurs', ...))
Elle vient de "$users" qui est définit juste au-dessus, ligne 807 :

$users = TableRegistry::get('LdapConnections')->getListUsers();
getListUsers() est définie dans src/Model/Table/LdapConnectionsTable.php

Attention, une fois qu'on choisit un utilisateur dans la liste, son email est cherché via le ldap pour l'afficher dans le champ suivant nommé "Email de l'utilisateur"
En plus, c'est un code javascript (ajax) qui fait ça à la fin du fichier add.ctp :

```
$("#nom-responsable").bind("change", function(event) {
    var url = document.URL;
    var reg = new RegExp("(materiels).*$", "g");
    var emailUrl = url.replace(reg, "Users/getLdapEmail/");
    $.ajax({
        url: emailUrl + $("#nom-responsable").val()
    }).done(function(data) {
        $("#email-responsable").val(data)
    });
});
```

Ce code appelle la fonction **getLdapEmail**() de **src/Controller/UsersController**.php

7.2. CRUD : Création, modification, suppression, et visionnage d'un utilisateur

(2/2/21 - EP)

On décrit ici quelles actions sont possibles sur les utilisateurs, en mode LDAP et en mode LOCAL (fake ldap) :

Action	mode LDAP	mode LOCAL (fake ldap)	Tables

(R)ead (view)	Oui (sauf password) *(profil superadmin only)*		users
(C)reate (add)	N/A	Oui, tous les attributs *(profil superadmin only)*	fakeldapusers et users
(U)pdate (edit)	Tous les attributs sauf nom, login, et email qui sont readonly *(profil superadmin only)*	Oui, tous les attributs *(profil superadmin only)*	fakeldapusers et users
(D)elete (delete)	N/A	Oui *(profil superadmin only)*	fakeldapusers et users

7.3. Authentification des utilisateurs (connexion avec ou sans LDAP, login)

Controller/ **UsersController** **login()**	**Controller/Component/** **LdapAuthComponent** **connection()**	**Model/Table/** **LdapConnectionsTable** **ldapAuthentication(** **$user_login, $user_password)**		
$user = $this->LdapAuth->**connection**()				
	return			

	TableRegistry::getTableLocator()->get('LdapConnections')->**ldapAuthentication**($login, $password);			
		// LDAP if ($this->LDAP_USED) { $filter = '(&'.$this->filter.'('.$this->authenticationType.'='.$user_login.'))'; $user_fetched = $this->_ldapSearch($filter, $just_these); if ($user_fetched['count'] != 0) { if ($this->_ldapAuth($user_fetched[0]["dn"], $user_password)) { $this->_saveNewUserInDB($user_fetched[0]); return $user_fetched[0]; } } return false; } // FAKE LDAP $user_fetched = $this->getFakeLdapUser($user_login); if ($user_fetched !== false) { // Est-ce le user FAKE LDAP par defaut (_fake_ldap_user_) ? si oui, on le laisse passer if ($user_fetched[DEFAULT_AUTH_TYPE][0] == $this->_getTheFakeLdapUser()['login'] && $user_fetched['userpassword'][0] == $this->_getTheFakeLdapUser()['pass']) return $user_fetched; // Sinon, on regarde si c'est un user de la table fakeldapusers if ((new DefaultPasswordHasher())->check($user_password,$ user_fetched['userpassword'][0])) return $user_fetched; }		

		return false		
if ($user) $this->LdapAuth->setUser($user) ; $this->statsUpdateForCurrentUse rWhen(null,'sur login'); return $this->redirect($this->LdapAuth-> redirectUrl());				

7.4. (plus détaillé) Authentification des utilisateurs (connexion avec ou sans LDAP, login)

(updated 02/02/2021 - EP)
\authentication \authentification \login \logout

(anglais : AUTHentication)

Nous décrivons ici le processus de CONNEXION d'un utilisateur avec un login et mot de passe. Il est possible AVEC ou SANS LDAP.

Ce processus se nomme en anglais "**authentication**". Pour plus d'information à ce sujet, lire la documentation de CakePhp (https://book.cakephp.org/3.0/en/controllers/components/authentication.html)

1) BOOTSTRAP

(Cette étape est décrite pour être exhaustif, mais vous pouvez aller directement à l'étape suivante, "2- LOGIN")

On décrit ici le chemin parcouru (dans l'ordre) depuis le tout début, avant d'arriver à la page de login.

Enchaînement des fichiers, depuis la racine du projet LABINVENT/ :

index.php

webroot/**index.php** :
```
require config/requirements.php
require vendor/autoload.php
new Application('config')
```

src/**Application.php**
```
class Application extends BaseApplication
    function bootstrap() {
        parent::bootstrap()
        addPlugin(Migrations)
        addPlugin(DebugKit)
```

vendor/cakephp/.../BaseApplication.php
```
function bootstrap() {
```

require_once config/bootstrap.php

config/**bootstrap.php**
require config/**paths.php**
require vendor/autoload.php
require vendor/cakephp/.../config/bootstrap.php
load config/**app.php** *(ce fichier contient ma config perso)*
set config (cache, db, email, log, …)

src/Controller/PagesController/display()

src/Controller/UsersController/login() sans POST

src/Template/Users/login.ctp => formulaire de login => POST

2) LOGIN (avec ou sans LDAP)

Une fois le formulaire de LOGIN “posté” (validé par l’utilisateur), on arrive à l’étape de login proprement dite.

Pour info, si ça bogue *(par exemple, à l’époque d’un fameux stagiaire..., on retournait "YOLO" au lieu de FALSE en cas d'échec de connection ldap, et ça plantait lamentablement, je m’en souviens très très bien...),*
*⇒ c'est *****src/Template/Error/error400.ctp***** qui prend le relais...*

Voici la description du processus de login (authentification) :

a) src/Controller/UsersController.php, fonction login() :

```
$user = $this->LdapAuth->connection();

// Utilisateur identifié
if ($user != FALSE) {
    $this->LdapAuth->setUser($user);
    // On va maintenant à la page qui était demandée
    return $this->redirect($this->LdapAuth->redirectUrl());
}

// Utilisateur non reconnu
```

```
$this->Flash->error(__('Login ou mot de passe invalide, réessayez'));
```

b) *src/Controller/Component/LdapAuthComponent.php, fonction connection() :*

```
// Get login and password entered by the current user (who is trying to connect)
$login = $this->request->getData('ldap');
$password = $this->request->getData('password');
return TableRegistry::getTableLocator()->get('LdapConnections')->ldapAuthentication($login, $password);
```

c) *src/Model/Table/LdapConnectionsTable.php, fonction ldapAuthentication($user_login, $user_password) :*

```
// Bad configuration => FAIL
if (! $this->_checkAndGetConfiguration()) return FALSE;

try {
    // REAL LDAP
    if ($this->LDAP_USED) {
        // No connexion allowed without password
        if (strlen(trim($user_password)) == 0) return FALSE;
        $just_these = [];
        $filter = "(&".$this->filter."(".$this->authenticationType . '=' . $user_login."))";
        $user_fetched = $this->_ldapSearch($filter, $just_these, $user_login, $user_password);
        if ($user_fetched !== FALSE) return $user_fetched[0];
    }

    // FAKE LDAP used
    else {
        $user_fetched = $this->getFakeLdapUser($user_login);
        /* Voici un exemple de ce qui est dans $user_fetched (fake ldap) :
        [
        'sn' => [
                    (int) 0 => 'Pallier'
                ],
```

```
                        'mail' => [
                                (int) 0 => 'Etienne.Pallier@irap.omp.eu'
                        ],
                        'givenname' => [
                                (int) 0 => 'Etienne'
                        ],
                        'uid' => [
                                (int) 0 => 'epallier'
                        ],
                        'userpassword' => [
                                (int) 0 => '<mot de passe crypté>'
                        ]
                ]
                */
                if ($user_fetched !== false) {

                        // Est-ce le user FAKE LDAP par defaut (_fake_ldap_user_) ? si oui, on le laisse passer
                        if ($user_fetched[$this->authenticationType][0] == $this->_getTheFakeLdapUser()['login'] &&
        $user_fetched['userpassword'][0] == $this->_getTheFakeLdapUser()['pass'])
                                return $user_fetched;

                         // Sinon, on regarde si c'est un user de la table fakeldapusers
                        if ( (new DefaultPasswordHasher())->check($user_password,$user_fetched['userpassword'][0]) )
                                return $user_fetched;
                }

        } // $this->LDAP_USED ?

} catch (Exception $e) {
        //echo 'Exception LDAP : ',  $e->getMessage(), "\n";
}

// Il y a eu un problème, l'utilisateur n'est pas reconnu
return FALSE;
```

d) 2 cas possibles :

- (1) si LDAP, on a appelé la fonction **_ldapSearch***($*filter, $just_these, $user_login, $user_password) *:*
- (2) si FAKE LDAP, on a appelé la fonction **getFakeLdapUser***($*user_login) *:*

Ces 2 fonctions retournent un user ($**user_fetched)**

Dans le détail :

- **Cas (1) (LDAP) : SEARCH en 4 étapse**

```
// (1) CONNEXION
$ldapConnection = ldap_connect($this->host, $this->port) or die("Could not connect to $this->host (port $this->port)");
if ($ldapConnection) {

        // (2) SET OPTIONS
        ldap_set_option($ldapConnection, LDAP_OPT_PROTOCOL_VERSION, 3);

        // (3) BINDING OPTIONNEL (true by default if not done)
        $ldapbind = TRUE;

        // - Authentified LDAP
        if ($this->ldap_authentified)
                $ldapbind = @ldap_bind($ldapConnection, $this->bindDn, $this->bindPass);

        // - Anonymous LDAP
        // (EP) En cas de LDAP anonyme, binding uniquement si login session (pour vérifier le mot de passe de l'utilisateur).
        // Car sans cette ligne, on passe avec n'importe quel password !!!
        else if ($user_login && $user_password)
                $ldapbind = @ldap_bind($ldapConnection, $this->authenticationType . '=' . $user_login . ',' . $this->baseDn, $user_password);

        // (4) SEARCH
        if ($ldapbind) {
                $results = ldap_search($ldapConnection, $this->baseDn, $filter, $just_these) or die(...);
                $search = ldap_get_entries($ldapConnection, $results);
                if ($search === FALSE) die(...);
                return $search;
```

```
            }

        }

        // Il y a eu un pb, utilisateur non reconnu
        return FALSE;
```

- **Cas (2) (FAKE LDAP) :**

e) On retourne maintenant à l'étape a) ci-dessus, à la suite de la ligne "$user = $this->LdapAuth->connection()"

7.5. Autorisations - Gestion des profils utilisateurs

\acl
(anglais : AUTHorizations)

Cette partie fait l'objet d'un document séparé

8. Base de Données

8.1. Accès à la BD LabInvent

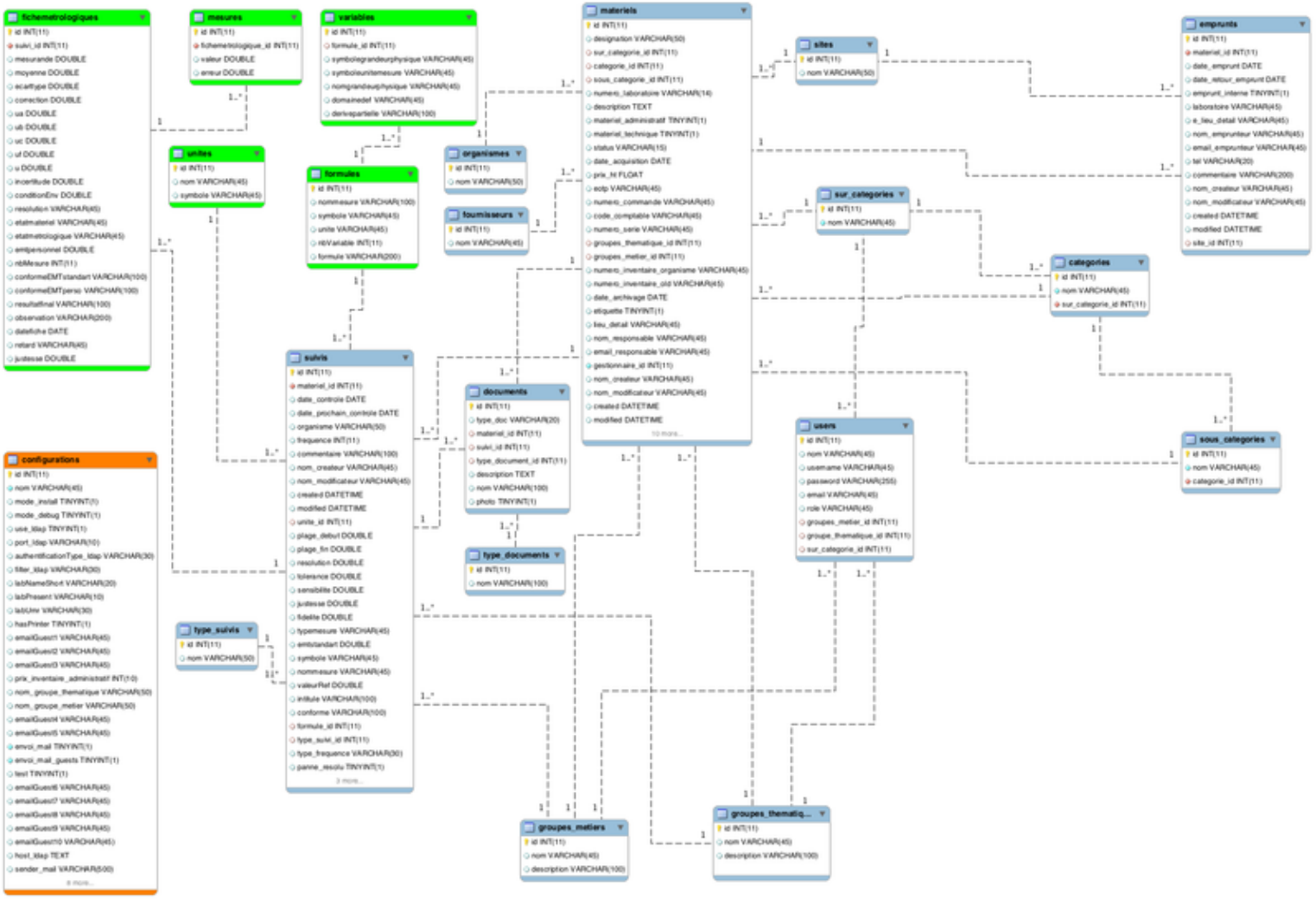
Vous pouvez accéder à la BD LabInvent de 2 façons :

- En **mode commande** (commande “**mysql**”) via les scripts CONNECT_TO_MYSQL_AS_USER ou CONNECT_TO_MYSQL_AS_ROOT (si l’installation a été faite avec docker, vous pouvez aussi utiliser directement les scripts DOCKER_LABINVENT_DB_USER ou DOCKER_LABINVENT_DB_ROOT)

- Via un client **PhpMyAdmin :**
    - si l’installation a été faite avec docker, et que vous avez installé l’environnement de dev, vous avez un phpmyadmin accessible à l’adresse http://localhost:8080 :
        - Accès seulement à la BD projet : user “labinvent_user”, pass “labinvent” (sauf si vous avez changé ces valeurs dans le fichier .env)
        - Accès à tout le serveur MySql : user “root”, pass “labinvent” (sauf si vous avez changé ces valeurs dans le fichier .env)
    - sinon, c’est à vous de mettre en place un phpmyadmin

8.2. Modèle de données (Data model)

Schéma de la base de données (v2.6) :



Légende:

- en orange : la partie "configuration" de labinvent ; c'est cette table qui contient tous les paramètres configurables du logiciel
- en vert : la partie "metrologie" qui est une extension de labinvent ajoutée par le laboratoire LATMOS ; cette extension est désactivée par défaut dans la configuration
- en bleu : tout le reste, c'est à dire la partie principale : les matériels, les suivis et les emprunts

9. Autres Descriptions techniques

Cette section décrit différents aspects internes du fonctionnement du logiciel.

Voici d’autres éléments plus anciens sur le wiki (à mettre à jour et intégrer dans cette présente doc) :

- Doc technique
- Doc de développement

9.1. Page d'accueil (démarrage, home page)

(updated 29/1/19 - EP)

LabInvent démarre sur la page **src/Template/Pages/home.ctp**

Cette page est la page d'accueil par défaut produite par le framework CakePhp (et est affichée tant que le mode debug est actif).

Elle a été modifiée pour LabInvent avec le contenu suivant :

```
<?php
if ($configuration->mode_install) {
	include ("home_install.ctp");
} else {
	include ("home_app.ctp");
}
?>
```

Comme on peut le voir, si on est en mode installation (lors de la phase d'installation du projet), on va sur la page **home_install**.ctp qui aide à vérifier si l'installation s'est bien passée ou bien s'il subsiste des problèmes à régler.

Sinon (situation par défaut, une fois l'installation terminée), on va sur la page **home_app**.ctp qui est la page d'accueil par défaut.

9.2. Ce qui se passe avant l'affichage d'une page web (workflow)

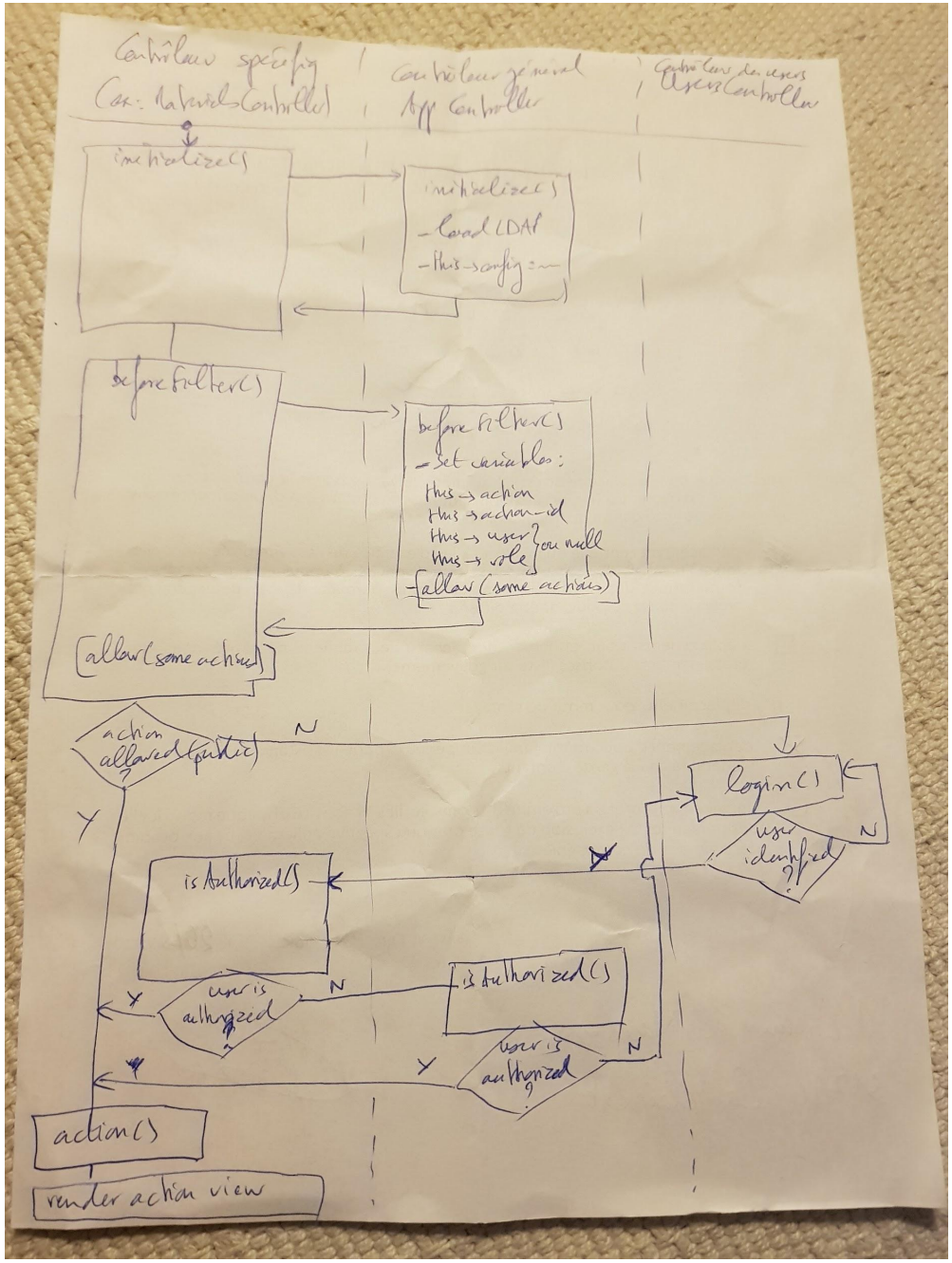
\authentication \authentification
\authorization \autorisation
\acl
(updated 28/4/20 - EP)
(TODO: 13/3/20 : à mettre à jour car j'ai changé la numérotation des steps, voir à la fin de ce chapitre, le point 4 - Exemples)

Cette section est un peu technique mais très utile pour comprendre dans quel ordre les méthodes des contrôleurs sont appelées, depuis l'appel d'une action, jusqu'à arriver finalement à l'affichage d'une vue (la page web qu'on voit au final sur le site).

Le mieux pour comprendre est de passer en mode DEBUG (Outils/Configuration générale de l'application/Editer/Cocher "Mode DEBUG").

Une fois ce mode DEBUG activé, vous verrez s'afficher les différentes étapes.

1 - Workflow général



Step 0 : initialize() - AUTHENTIFICATION

- S'il n'existe pas de méthode initialize() dans le contrôleur spécifique appelé, c'est celle de AppController qui est appelée
- S'il existe une méthode initialize() dans le contrôleur spécifique appelé, alors c'est celle-ci qui est appelée d'abord. Cette méthode doit explicitement commencer par appeler la méthode initialize() de AppController (via une instruction "parent::initialize()")
- Si cette méthode autorise des actions avec l'instruction $this->LdapAuth->**allow** (ou Auth->allow), toutes ces actions sont accessibles SANS authentification (elles sont PUBLIQUES, pas besoin de login) ; ex: $this->LdapAuth->allow(['display', 'view', 'index']);

Step 1 : beforeFilter() - AUTHENTIFICATION

- Cette méthode est appelée implicitement par initialize()
- Même principe que pour l'étape précédente
- C'est à ce niveau par exemple qu'on autorise l'action display() (du controleur PagesController)

Step 2 : isAuthorized() - AUTHORIZATION

- ATTENTION, **cette méthode n'est appelée que APRES la connexion** (login, authentification), pas avant. Donc, si on n'est pas authentifié et que l'action demandée n'a pas été autorisée par allow() dans initialize() ou dans beforeFilter(), cette méthode **n'est PAS APPELÉE**
- **Donc, si on est authentifié, alors cette méthode est appelée selon le même principe que dans les étapes précédentes**
- **C'est cette méthode qui gère l'aspect AUTHORIZATION :**
    - **si elle "return true", l'action en cours est autorisée**
    - **si elle "return false", l'action en cours est interdite**

2 - Synthèse :

Méthode (fonction)	AppController (initialisations générales)	un controleur spécifique **(ex: MaterielsController)** (initialisations spécifiques)
A - A ce niveau, l'utilisateur n'est pas encore authentifié (non connecté, non logué) *On va donc définir les actions publiques éventuelles qui sont autorisées sans connexion*		
(0) **initialize()**	*// Initialisation du composant d'authentification LdapAuth* *// (ou Auth si on n'utilise pas Ldap)* $this->**loadComponent('LdapAuth'**, …) *// Récupération de la config* $this->confLabinvent = TableRegistry::getTableLocator()->**get('Configurations'**)->find()->first()	*// On pourrait ici faire des trucs spécifiques, mais c'est pas le cas…* *// Appel de la méthode initialize() de AppController* parent::**initialize**()
(1) **beforeFilter($event)**	// 1) On définit des variables générales qu'on va passer systématiquement à TOUTES les **vues** $var1 = … $var2 = … ... $this->**set**(compact('var1', 'var2', ...)) *// 2) Mais surtout, on autorise certaines actions SANS connexion (ces actions sont donc publiques)* *// - pour le mode spécial "installation", on autorise pleins d'actions sans login* // - pour le mode normal habituel, on autorise seulement l'action "display" pour quelques pages publiques (comme la page "about") if ($configuration->mode_install) $this->LdapAuth->**allow**(['display', 'add', 'edit', 'installOff']); else $this->LdapAuth->**allow**(['display']);	*// On appelle d'abord la méthode initialize() de AppController* *// pour les autorisations générales* parent::**beforeFilter**($event) *// On pourrait ici faire des trucs (ou autorisations) spécifiques, mais c'est pas le cas pour l'instant…*
B - A ce niveau, l'utilisateur est maintenant authentifié ***On va donc maintenant définir les actions qui sont autorisées (après connexion)***		
(2) **isAuthorized($user)**	*// On tente d'autoriser ici les actions qui n'ont pas été autorisées par la méthode* isAuthorized du controleur spécifique *// On récupère les paramètres nécessaires :* $role = $this->getUserRole($user) $action = $this->getActionPassed() *// On autorise ou pas l'action demandée :* *// - Super-Admin peut accéder à toutes les actions* if ($role == 'Super Administrateur') **return true**;	*// On définit ici les actions autorisées ou pas par ce contrôleur spécifique* *// On récupère les paramètres nécessaires :* *// - ex: 'uid'* $this->userCname = $user[$configuration-> ldap_authenticationType][0] *// - le nom de l'action demandée...* $action = $this->getActionPassed() *// - l'id éventuel (ex: .../view/3)*

	// - Actions accessibles à TOUS les roles (profils), quelque soit le controleur if (in_array($action, ['index', 'view', 'add', 'find', 'creer', 'getNextDate', 'getDateGarantie'])) **return true**; *// - Pour toutes les autres actions => accès refusé (denied)* **return false**;	$id = $this->getIdPassed() *// On autorise ou pas l'action demandée :* *// - SUPERADMIN : par défaut il a tous les droits* *// (sauf action 'edit' si matos validé, pour faire comme pour ADMIN),* *// donc on autorise l'action demandée => return true* if ($this->userRole=='Super Administrateur' && $action!='edit') **return true** *// - Autres cas : selon les contrôleurs, on **return :*** *// - soit **isAuthorizedAction**($this->userRole,$action, $id,$user,$this->userCnam) si on est dans MaterielsController (cette fonction a été développée spécialement pour le contrôleur de Materiels car il y a bcp d'actions et des cas complexes à traiter ; elle retournera AppController.isAuthorized() à la fin)* *// - soit AppController.**isAuthorizedCommons**(), qui retournera aussi AppController.isAuthorized() à la fin* *// - soit directement (et uniquement) AppController.**isAuthorized**()*
Cas unique du MaterielsController **:** **isAuthorizedAction**($role, $action, $id, $user, $userCname)	*cette fonction n'existe pas dans AppController*	*// Pour chaque action de ce contrôleur, on return true ou false selon qu'on l'autorise ou pas* switch ($**action**) { case 'add': … return... case 'edit': … return... case 'view': … return... case 'index': … return... case 'delete': … return... } *// Si aucune règle ci-dessus n'a return true (ou false)* *// => DEFAULT PARENT RULE* *// (on appelle la méthode isAuthorized() de AppController)* **return** parent::**isAuthorized**($user);
Cas de la plupart des autres contrôleurs : **isAuthorizedCommons** ($user)	*// Seul Administration (et +) peut ajouter, supprimer ou modifier (pour la plupart des controleurs) :* *$action = $this->getActionPassed();* *if (in_array($action, **['add', 'edit', 'delete']**)) {* ***return** ($this->USER_IS_ADMIN_AT_LEAST());* *// Sinon, on applique les règles générales par défaut* *return AppController::**isAuthorized**($user);*	

3 - Dans le détail :

1 - Si vous n'êtes pas encore connecté, vous verrez s'afficher, juste au-dessus de la page d'accueil (login), dans l'ordre :

'step AVANT 0: AppController.**initialize**()'

'step 0: AppController.**beforeFilter**()'

'step 2: **UsersController.login**()'

'step 3: AppController.**beforeRender**()'

2 - Une fois connecté (logué), c'est comme si vous aviez appelé l'action "**display**" du controleur "**PagesController"** (controleur responsable de l'affichage des pages webs classiques), et donc vous verrez s'afficher, juste au-dessus de la page d'accueil (après authentification), dans l'ordre :

'step AVANT 0: AppController.**initialize**()'

'step 0: AppController.**beforeFilter**()'

'step 2: **PagesController**.**display**()'

'step 3: AppController.**beforeRender**()'

3 - Si maintenant vous affichez la liste des matériels (action **"index"** du controleur **"MaterielsController**"), vous verrez s'afficher, juste au-dessus de cette liste, la séquence complète, dans l'ordre :

'step AVANT 0: AppController.**initialize**()'

'step 0: MaterielsController.**beforeFilter**()' ⇒ **ce qui va appeller** ⇒ 'step 0: AppController.beforeFilter()'

'step 1: MaterielsController.**isAuthorized**()'

'step 2: **MaterielsController**.**index**()'

'step 3: MaterielsController.**beforeRender**()' ⇒ **ce qui va appeller** ⇒ 'step 3: AppController.beforeRender()'

Quelques explications s'imposent. Les méthodes suivantes sont appelées dans cet ordre :

- src/Controller/**AppController.initialize**() :
    - Cette **méthode est appelée en tout premier lieu**
    - on va notamment y lire (une fois pour toutes) la **configuration** et la mettre dans une variable $**this->confLabinvent**, utilisée partout dans le code ensuite

- src/Controller/**AppController.beforeFilter**() :
    - cette méthode est appelée (après initialize()) par défaut quand on ne sait pas quel controleur est responsable de l'action, ou bien APRES l'appel de la méthode éponyme d'un controleur spécifique (par exemple, MaterielsController.beforeFilter() est appelée, ce qui va appeler ensuite AppController.beforeFilter())
    - on y initialise (une fois pour toutes) des variables utilisées partout dans le code, telles que le profil de l'utilisateur connecté (ce qui va déterminer ses droits ensuite) :
        - $this->USER_IS_UTILISATEUR
        - $this->USER_IS_RESPONSABLE
        - $this->USER_IS_ADMIN
        - $this->USER_IS_ADMINPLUS
        - $this->USER_IS_SUPERADMIN
        - $this->USER_IS_RESPONSABLE_OR_MORE
        - $this->USER_IS_ADMIN_OR_MORE
        - $this->USER_IS_ADMINPLUS_OR_MORE
    - on passe aussi ces variables à la VUE grâce à la méthode $this->set() ; toutes les VUES vont donc hériter automatiquement de ces variables et pourront les utiliser directement ainsi :
        - if ($USER_IS_UTILISATEUR) …
        - if ($USER_IS_ADMIN) ...

- src/Controller/MaterielsController.**isAuthorized**() :
    - Si une méthode isAuthorized() existe dans le controleur spécifique responsable de l'action demandée (par exemple MaterielsController si l'action est "afficher la liste des materiels", c'est à dire, materiels/index), elle est appelée pour savoir si l'action demandée est autorisée ou pas pour l'utilisateur courant. Cette méthode retourne VRAI si l'action est autorisée ou FAUX sinon. Si FAUX, alors on ne va pas plus loin et un message du genre "cette action n'est pas autorisée" va s'afficher en haut de page.

- src/Controller/MaterielsController.**index**() :

- C’est maintenant la méthode portant le nom de l’action demandée qui est appelée sur le controleur compétent. Ici, par exemple, on appelle la méthode index() du controleur MaterielsController si on a demandé la liste des matériels (action “materiels/index”).
- Dans cette méthode (correspondant à une action), on va initialiser des variables qui seront passées à la vue correspondant à cette action (ici la vue src/Template/Materiels/index.ctp), grâce à la méthode $this->set(). Ces variables pourront être directement utilisées dans le fichier de la vue (voir ci-après). Par exemple, dans l’action “index()” du controleur MaterielsController, on va initialiser une variable $**materiels** qui contient la liste des matériels à afficher.

- src/Controller/MaterielsController.**beforeRender() :**
  - Juste avant l’affichage de la vue (ici la vue index, voir ci-dessous), il y a encore la méthode beforeRender() qui est appelée, et qu’on peut utiliser pour définir quelques variables générales pour la vue, mais il est préférable de le faire dans **beforeFilter**() (voir ci-dessus).
  - Actuellement, on utilise cette méthode pour initialiser des FONCTIONS utilitaires dont TOUTES les vues vont héritées (encore grâce à la méthode $this->set()) ; On définit ainsi les fonctions $**displayElement**(), $**dateProchainControleVerif**(), et $**echoActionButton**(), très utilisées par les vues pour afficher des boutons, ou autres éléments...

- src/Template/Materiels/**index.ctp** :
  - On est maintenant à la fin du workflow, sur la dernière page exécutée, la vue. Ici, il s’agit de la vue “index.ctp” (du template Materiels) qui affiche la liste des materiels. Comme cette vue a hérité de la variable $materiels (voir ci-dessus), elle n’a plus qu’à boucler sur cette liste pour afficher les materiels un par un. Elle hérite aussi des variables générales initialisées dans **AppController.beforeFilter()** telles que $USER_IS_UTILISATEUR… (voir ci-dessus) pour décider de ce qu’elle peut afficher ou pas.
  - De manière générale, **la vue doit être la plus BETE possible**, et se contenter d’afficher des éléments un par un, sans avoir trop à réfléchir car presque toute la réflexion doit avoir été faite en amont, dans le controleur. **Ca doit donc être du style** “Si l’utilisateur courant est un ADMIN, alors je peux afficher les informations administratives”, **mais pas du style** “Si l’utilisateur courant est un ADMIN, et si le matériel est VALIDATED ou alors Si l’utilisateur est un RESPONSABLE et qu’il est lié à la thématique du matériel en cours… etc., alors je peux afficher les informations suivantes...”

4 - Exemples

(EP updated 13/3/20 => nouvelle numérotation des steps)
(Activer d’abord le mode DEBUG)

- **fournisseurs/index**

Cette URL va engendrer ces étapes :

'step 0: AppController.initialize()'

'step 1B (general): AppController.beforeFilter()'
'- userRole is Super Administrateur'
'- profile is: 5'

'step 2A (specific): FournisseursController.isAuthorized(user)'

'step 2B (intermediaire general): AppController.isAuthorizedCommons(user)'
'- action is: index'
'step 2C (general): AppController.isAuthorized()'
'- role is Super Administrateur'

'step 4B (general) : AppController.beforeRender() - [step 3 = action() si existe]'

- **/materiels/index**

Cette URL va engendrer ces étapes :

'step 0: AppController.initialize()'

'step 1A (specific): MaterielsController.beforeFilter()'
'step 1B (general): AppController.beforeFilter()'
'- userRole is Super Administrateur'
'- profile is: 5'

'step 2A (specific): MaterielsController.isAuthorized(user)'

'step 3: MaterielsController.index()'

'step 4A (specific) : MaterielsController.beforeRender() - [step 3 = action() si existe]'
'step 4B (general) : AppController.beforeRender() - [step 3 = action() si existe]'

- **materiels/view/12006**

Cette URL va engendrer ces étapes :

'step 0: AppController.initialize()'

'step 1A (specific): MaterielsController.beforeFilter()'
'step 1B (general): AppController.beforeFilter()'
'- userRole is Super Administrateur'
'- profile is: 5'

'step 2A (specific): MaterielsController.isAuthorized(user)'

'step 3: MaterielsController.view()'

'step 2B (intermediaire): MaterielsController.isAuthorizedAction(Super Administrateur, edit, 12006, user, epallier)'

'step 4A (specific) : MaterielsController.beforeRender() - [step 3 = action() si existe]'
'step 4B (general) : AppController.beforeRender() - [step 3 = action() si existe]'

- **materiels/edit/12007**

Cette URL va engendrer ces étapes :

'step 0: AppController.initialize()'

'step 1A (specific): MaterielsController.beforeFilter()'
'step 1B (general): AppController.beforeFilter()'
'- userRole is Super Administrateur'
'- profile is: 5'

'step 2A (specific): MaterielsController.isAuthorized(user)'
'- user is:'

'step 3: MaterielsController.add_or_edit()'

'now :2020-03-13 17:16:54'
'cached :13/03/2020 12:11'

'cached2 :13/03/2020 12:11'
'Temps écoulé depuis last save:'
'5 mn 54 sec'
...

'step 4A (specific) : MaterielsController.beforeRender() - [step 3 = action() si existe]'
'step 4B (general) : AppController.beforeRender() - [step 3 = action() si existe]'

9.3. LOG

(updated 19/12/18 - EP)

Des messages de debug et d'erreur peuvent être loggés grâce au système de logging de CakePhp qui est configuré dans le fichier de configuration générale **config/app.php** :

```
/**
 * Configures logging options
 */
'Log' => [
        'debug' => [
                'className' => 'Cake\Log\Engine\FileLog',
                'path' => LOGS,
                'file' => 'debug',
                'levels' => ['notice', 'info', 'debug'],
                'url' => env('LOG_DEBUG_URL', null),
        ],
        'error' => [
                'className' => 'Cake\Log\Engine\FileLog',
                'path' => LOGS,
                'file' => 'error',
                'levels' => ['warning', 'error', 'critical', 'alert', 'emergency'],
                'url' => env('LOG_ERROR_URL', null),
        ],
],
```

Remarque :

error_log("bla bla bla");
==> écrit dans **logs**/error.log

Dans une ancienne version de LabInvent, on pouvait faire ceci (il faudrait remettre ça en place) :

```
$this->log('Something broke');
```

==> écrit dans cakephp/app/tmp/logs/**labinvent.log**

9.4. DEBUG

BUT : CORRIGER UNE ERREUR, RESOUDRE UN PROBLEME, COMMENT DEBOGUER (DEBUG)...

Aller à la section consacrée au mode DEBUG.

9.5. Validation des données entrées par formulaire

\valid
(17/9/20 - EP)

https://book.cakephp.org/4/en/orm/validation.html

Comment les données entrées via formulaire (par exemple une fiche de matériel) sont-elles validées ?

Elles le sont en 2 temps :

1) **Validation du format** des données
   Ex:
   - vérifier le format d'une date
   - vérifier le format d'un email
   - ...
2) **Validation de la cohérence** des données saisies par rapport aux autres données de l'entité (ex: matériel), ou aux autres entités
   Ex :
   - vérifier que la date de livraison d'un matériel n'est pas dans le futur, et qu'elle est postérieure à la date d'achat, mais pas trop non plus
   - vérifier que la personne sélectionnée pour être le gestionnaire du matériel existe et a bien un profil d'administratif
   - s'assurer qu'un email est unique
   - ...

9.5.1. Synthèse des 2 étapes

Pour le détail de chaque étape, continuer la lecture plus loin.

<table>
<tr><th></th><th>Moment du déclenchement de la validation</th><th>Lieu de la définition des règles de validation</th></tr>
<tr><td colspan="3">Etape 0 - L’entité n’existe pas encore, on a juste un tableau des données saisies dans le formulaire (string, int, float…)</td></tr>
<tr><td>Etape 1 - Validation du format, syntaxe</td><td>newEntity ou patchEntity :

$e = $articles->newEntity(
$this->request->getData()
);
if ($article->getErrors()) {
// Entity failed validation.
}</td><td>Table des entités, fonction validationDefault() :

class ArticlesTable extends Table {
public function validationDefault(Validator $validator): Validator {
$validator
->requirePresence('title', 'create')
->notEmptyString('title');
$validator
->allowEmptyString('link')
...

return $validator;
}</td></tr>
<tr><td colspan="3">Etape 1b - L’entité existe, les champs sont donc typés (ex: un champ date n’est plus une string, mais un objet DateTime)</td></tr>
<tr><td>Etape 2 - Validation de la cohérence</td><td>save() ou delete() :

if (! $articles->save($e)) {
// Erreur de validation
$e->getErrors();
$e->getError(‘email’);
}
else {
// La sauvegarde s’est bien passée</td><td>Table des entités, fonction buildRules() :

class ArticlesTable extends Table {

public function buildRules(RulesChecker $rules): RulesChecker {
$rules->add(function ($entity, $options) {
// Return a boolean to indicate pass/failure
}, 'ruleName');
$rules->addCreate(function ($entity, $options) {
// Return a boolean to indicate pass/failure</td></tr>
</table>

	... }	}, 'ruleName'); … **return $rules**; }

Le détail des étapes 1 et 2 est donné ci-dessous.

9.5.2. Etape 1 - Validation du format, syntaxe

L'étape 1 est faite au moment où on crée une Entité (**Entity** $e) à partir des données reçues du formulaire (récupérées via **getData**()), à l'aide des fonctions **newEntity**() ou **patchEntity**() de la classe Table :

```
$e = $articles->newEntity($this->request->getData());
if ($article->getErrors()) {
    // Entity failed validation.
}
```

Pour désactiver la validation :

```
$e = $articles->newEntity(
    $this->request->getData(),
    ['validate' => false]
);
Idem avec patchEntity :
$e = $articles->patchEntity( $e, $newData, [ 'validate' => false] );
```

To create a **default validation object** in your table, create the **validationDefault**() function:

```
class ArticlesTable extends Table
{
    public function validationDefault(Validator $validator): Validator {
        $validator
            ->requirePresence('title', 'create')
```

```
            ->notEmptyString('title');

        $validator
            ->allowEmptyString('link')
            ->add('link', 'valid-url', ['rule' => 'url']);

            // Utilisation d'une fonction de validation custom :
            $validator
                    ->add($f, 'valide1', [
                            'rule' => 'dateIsValid',
                            'message' => "La date n'est pas valide (JJ/MM/AAAA)",
                            'provider' => 'table',
                    ]);

        ...

        return $validator;
    }
}

// Définition des fonctions de validation custom :
public function dateIsValid($value, array $context) {
        ...
}
```

A ce niveau de validation, on peut voir qu'on traite les champs du formulaire tels qu'ils sont retournés par la fonction this->request->**getData(), c'est à dire** des champs textes (string), entiers, ou float… Même les dates sont retournées en tant que "string".
Les fonctions de validation custom reçoivent 2 paramètres :

- $**value** : la valeur du champ à valider (souvent une string)
- $**context** : tous les autres champs
    - $context->**newRecord** est un boolean
    - $context->**data** est l'entité complète, sous forme d'un tableau qui contient tous les champs (souvent des 'string')

9.5.3. Etape 2 - Validation de la cohérence

On appelle ces règles de validation les **règles de domaine ou d'application** (domain/application rules), ou encore les **règles métier**.

Ces règles sont appliquées au moment où on sauvegarde ou on supprime l'entité, avec save() et delete() :

```
if (! $articles->save($e)) {
        // Erreur de validation
        $e->getErrors(); // Contains the domain rules error messages
        $e->getError('email'); // pour récupérer seulement l'erreur sur le champ 'email'
}
else {
        // La sauvegarde s'est bien passée
        ...
}
```

Pour désactiver cette validation :

```
$articles->save($article, ['checkRules' => false]);
```

Ces règles sont définies dans la classe de la Table, avec la fonction **buildRules() :**

```
public function buildRules(RulesChecker $rules): RulesChecker {
    // Add a rule that is applied for create and update operations
    $rules->add(function ($entity, $options) {
        // Return a boolean to indicate pass/failure
    }, 'ruleName');

    // Add a rule for create.
    $rules->addCreate(function ($entity, $options) {
        // Return a boolean to indicate pass/failure
    }, 'ruleName');

    // Add a rule for update
    $rules->addUpdate(function ($entity, $options) {
        // Return a boolean to indicate pass/failure
    }, 'ruleName');
```

```
    // Add a rule for the deleting.
    $rules->addDelete(function ($entity, $options) {
        // Return a boolean to indicate pass/failure
    }, 'ruleName');


    return $rules;
}
```

9.6. Raccourcis d’écriture, variables globales et fonctions (méthodes) pratiques pour les développeurs

\raccourci \shortcut

- **dans AppController (donc disponible pour TOUS les controleurs) :**
  - this->**SUPERADMIN_CAN_DO_EVERYTHING** : mettre à true pour donner rapidement TOUS les droits à superadmin (debug only)
  - $this->**u** = user courant
  - $this->**e** = entité courante
  - $this->**e_id** = id de l'entité courante (a priori égale à $this->e->id)
  - $this->**a** = action en cours

- $this->**c** = controleur en cours (?)
- $this->**USER_IS_USER**()
- $this->**USER_IS_RESP**()
- $this->**USER_IS_ADMIN**()
- $this->**USER_IS_ADMINPLUS**()
- $this->**USER_IS_SUSERADMIN**()

// méthodes optimisées : le matériel (entité) et/ou le user ne sont lus dans la BD que si pas déjà chargés
- $this->**getEntity**($id=null) // id=null si matos courant
- $this->**getUserByLogin**($userlogin)
- $this->**getUserByFullname**($fullname)

- **dans MaterielsController (contrôleur le plus important de tous) :**
  - $this->**isCreated**($id=null) // id=null si matos courant
  - $this->**isValidated**($id=null)
  - $this->**isTobearchived**($id=null)
  - $this->**isArchived**($id=null)

  // méthodes optimisées : le matériel et le user ne sont lus dans la BD que si pas déjà chargés
  - $this->**belongsToUser**($username, $id=null) // id=null si matos courant
    et $this->**belongsToCurrentUser**($id=null)
  - $this->**isSameGroupAsUser**($userlogin, $id=null) // id=null si matos courant
    et $this->**isSameGroupAsCurrentUser**($id=null)

- **dans entité Materiel (Entity) (entité importante) :**
  - $this->**is_created**, $this->**is_validated**, $this->**is_tobearchived**, $this->**is_archived**
  - $this->**belongsToUser**($username) // owned or declared by user
  - $this->**isSameGroupAsUser**($usergroup1, $usergroup2) // is same group as one of user groups
  - …
- **dans entité User (Entity) (entité importante) :**
  - **$this->user_is_user**
  - **$this->user_is_resp**
    $this->user_is_admin
    $this->user_is_adminp

$this->user_is_super

10. HOWTO

\howto
(updated 8/9/20 - EP)

Ce document est un HOWTO, c'est à dire un guide technique pour savoir comment faire quoi.

Il donne des solutions à différents types de besoins ou problèmes, et explique aussi où trouver quoi.

Chaque recette donnée ici est qualifiée avec un niveau “**F**”acile, “**M**”oyen, ou “**D**”ificile.

10.1. Ajouter une nouvelle table en BD (ajouter une nouvelle Entité)

(updated 8/9/20 - EP)
Niveau : D

On explique ici ce qu'il faut faire quand on veut ajouter une nouvelle table dans la base de données,
table qui doit être prise en compte dans l'application en tant qu'Entity (objet entité).
Cette explication est donnée à travers 3 exemples, le premier étant le plus récent donc le plus pertinent.

Attention, après avoir ajouté une nouvelle table, pour qu'elle soit prise en compte par CakePhp, il faut absoluement supprimer le cache des modèles avec l'instruction suivante, depuis la racine du projet :

```
$ rm -rf tmp/cache/models/*
```

(sinon, vous allez avoir un comportement bizarre sur les vues !!!)

10.1.1. Premier exemple : ajout de la table projets (EP)

(updated 8/9/20 - EP)

a) Impact sur la BD

```
-- Creation de la table projets
CREATE TABLE projets (
  id int(11) PRIMARY KEY NOT NULL AUTO_INCREMENT,
  nom varchar(45) NOT NULL UNIQUE,
  description text NULL,
  groupes_thematique_id int(11) NULL,
  chef_science_id int(11) NULL,
  chef_projet_id int(11) NULL,
  date_start date NULL,
  date_stop date NULL
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
-- Si on voulait plutôt pointer les FK sur users(username) au lieu de users(id) :
-- chef_science_id varchar(45) NULL
-- chef_projet_id varchar(45) NULL
```

-- Ajout des contraintes d'unicité sur la (les) clé de la table users : nom, mais aussi username (ca serait mieux que nom)
-- Dans tous les cas, cette contrainte d'unicité se justifie
-- Mais elle est aussi NECESSAIRE si on veut que ces champs soient des clés (primaires) de la table users et que des FK puissent pointer dessus
ALTER TABLE users ADD UNIQUE(`username`);
ALTER TABLE `users` ADD UNIQUE(`nom`);

-- Ajout des contraintes FK pour cette table
ALTER TABLE **projets**
ADD CONSTRAINT fk_projets_groupes_thematique_id FOREIGN KEY(**groupes_thematique_id**) **REFERENCES** groupes_thematiques(id) on DELETE **set null**,
ADD CONSTRAINT fk_projets_chef_science_id FOREIGN KEY(**chef_science_id**) **REFERENCES** users(id) on DELETE no ACTION,
ADD CONSTRAINT fk_projets_chef_projet_id FOREIGN KEY(**chef_projet_id**) **REFERENCES** users(id) on DELETE no ACTION;
-- TODO: on pourrait aussi utiliser le champ username au lieu de id (modif future ?) :
-- ADD CONSTRAINT fk_projets_chef_science_id FOREIGN KEY(**chef_science_id**) **REFERENCES** users(username) on DELETE no ACTION,
-- ADD CONSTRAINT fk_projets_chef_projet_id FOREIGN KEY(**chef_projet_id**) **REFERENCES** users(username) on DELETE no ACTION;

((
ALTER TABLE projets DROP FOREIGN KEY fk_projets_chef_science_id;
ALTER TABLE projets DROP INDEX fk_projets_chef_science_id;
ALTER TABLE projets DROP FOREIGN KEY fk_projets_chef_projet_id;
ALTER TABLE projets DROP INDEX fk_projets_chef_projet_id;
))

-- Ajout dans la table materiels d'une FK vers cette nouvelle table
ALTER TABLE materiels
ADD projet_id INT(11) NULL after groupes_metier_id,
ADD CONSTRAINT fk_materiels_projet_id FOREIGN KEY (projet_id) REFERENCES projets(id) on DELETE **set null;**

b) <u>Impact sur les Modèles (cakephp/app/Model)</u>

On va ici ajouter essentiellement 2 classes de modèle qui vont permettre de gérer un projet comme un objet :

- **Model/Table/ProjetsTable :** représente la table et toutes ses lignes

- **Model/Entity/Projet :** représente une ligne de la table, c'est à dire un objet Projet (une instance)

Autant utiliser l'utilitaire "bake model" de CakePhp plutôt que de tout faire à la main :

$ **bin/cake bake model Projets**

- *Baking **table class** for Projets...*

*Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/**Model/Table/ProjetsTable**.php*

- *Baking **entity class** for Projet...*

*Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/**Model/Entity/Projet**.php*

- *Baking **test fixture** for Projets...*

*Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/**tests/Fixture/ProjetsFixture**.php*

- *Baking **test case** for App\Model\Table\ProjetsTable ...*

*Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/**tests/TestCase/Model/Table/ProjetsTableTest**.php*

Voir tous ces fichiers ci-dessus (J'ai aussi fait quelques petites retouches à la main)

c) Impact sur les Contrôleurs (cakephp/app/Controller)

On va ici créer le contrôleur des projets, c'est à dire la classe **Controller/ProjetsController.**

Autant utiliser l'utilitaire "bake controller" de CakePhp plutôt que de tout faire à la main :

$ bin/cake **bake controller Projets**

- *Baking controller class for Projets…*
  *Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/**Controller/ProjetsController**.php*
- *Baking test case for App\Controller\ProjetsController …*
  *Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/**tests/TestCase/Controller/ProjetsControllerTest**.php*

Voir tous ces fichiers ci-dessus (J'ai aussi fait quelques petites retouches à la main)

d) impact sur les vues (cakephp/app/View)

On va ici créer les vues 'index', 'view', 'add' et 'edit' des projets, permettant à la fois de créer, visualiser, et modifier les projets

Autant utiliser l'utilitaire "bake controller" de CakePhp plutôt que de tout faire à la main :

$ bin/cake **bake template Projets**

- Baking `**index**` view template file…
  Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/Template/Projets/index.ctp
- Baking `**view**` view template file…
  Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/Template/Projets/view.ctp
- Baking `**add**` view template file…
  Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/Template/Projets/add.ctp
- Baking `**edit**` view template file…
  Creating file /Users/epallier/_PROJ/_W/LABINVENT/SOURCE/labinvent2_DEV/src/Template/Projets/edit.ctp

e) (Optionnel) Regénérer les TESTS

Avec "bake", c'est facile de générer des squelettes de tests :

$ bin/cake bake test <type> <name>

<type> doit être une de ces options:

1. Entity
2. Table
3. **Controller**
4. Component
5. Behavior
6. Helper
7. Shell
8. Cell

f) Ajouter une entrée au menu pour les Projets (View/Pages/tools-sm)

g) impact sur les vues de Materiel

h) Supprimer le cache des modèles pour qu'il soit régénéré

Attention, pour que la nouvelle table soit bien prise en compte par CakePhp,
il faut absolument supprimer le cache des modèles avec l'instruction suivante, depuis la racine du projet :

```
$ rm -rf tmp/cache/models/*
```

(sinon, vous allez avoir un comportement bizarre sur les vues !!!)

10.1.2. Deuxième exemple : ajout de la table sur-categorie (EP)

(updated 4/2/19 - EP)

Avant cet ajout, il n'y avait que la table categorie et la table sous-categorie.
Vous trouverez plus de détails sur ce sujet dans la section suivante nommée "COMMENT J'AI FAIT POUR AJOUTER UN 3eme niveau de catégorie appelée sur_categorie (ou domaine)"

a) impact sur la BD

- ajout d'une nouvelle table sur_categories(id,nom) :

```
CREATE TABLE IF NOT EXISTS sur_categories (
  id int(11) NOT NULL AUTO_INCREMENT,
  nom varchar(45) DEFAULT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY nom_UNIQUE (nom)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

- ajout dans la table categories d'une cle etrangere sur_categorie_id :

```
ALTER TABLE categories
    ADD sur_categorie_id INT( 11 ) NULL DEFAULT NULL,
      ADD CONSTRAINT fk_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

- ajout dans la table materiels d'une cle etrangere sur_categorie_id :

```
ALTER TABLE materiels
    ADD sur_categorie_id INT( 11 ) NOT NULL after designation,
    ADD CONSTRAINT fk_materiels_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

(attention, il faut d'abord vider les lignes des tables suivis, emprunts, et materiels)
delete from suivis;
delete from emprunts;
delete from materiels;

Il faut aussi ajouter des sur-categories :
Ajout de quelques sur-categories
insert into sur_categories (id,nom) values
(1,'Electronique'),
(2,'Informatique'),
(3,'Instrumentation')
(4, 'Logistique'),
(5, 'Mecanique'),
(6, 'Optique')
;

Relier toutes les categories a une sur-categorie (la 1)
update categories set sur_categorie_id=1 where id<10;
update categories set sur_categorie_id=2 where id>=10 and id<20;
update categories set sur_categorie_id=3 where id>=20;

b) impact sur les modeles (cakephp/app/Model)
- Ajout du nouveau modèle SurCategorie.php : copie de Categorie.php avec "hasMany categorie"
- Modif du modèle Categorie.php : + belongsTo="SurCategorie"

c) impact sur les controleurs (cakephp/app/Controller)
- Ajout du nouveau controleur SurCategoriesController.php : minimaliste (comme CategoriesController)
- Modif du controleur CategoriesController.php : +getBySurCategorie()

d) impact sur les vues (cakephp/app/View)
- SurCategorie : no view (scaffold ?)
- Categorie : actuellement no view (scaffold ?)
--> creer une vue (comme SousCategorie)

+ get_by_surcategorie.ctp
+ scaffold.form.ctp

e) View/Pages/tools.ctp : ajouter une entree au menu pour les Sur-Categories

f) impact sur TOUTES les vues de Materiel
add/edit/view/index
dans la vue index, remplacer la categorie par la sur-categorie
GROS BOULOT sur la vue ADD/EDIT (+ javascript)

10.1.3. Troisième exemple : ajout de la table type_suivis (VM)

(updated 4/2/19 - EP)

Un peu plus haut est expliqué comment créer une table, je vais ici expliquer comment créer et remplir tout ce qui est basiquement nécessaire pour l'utiliser, en prenant pour exemple la table type_suivis.

Après avoir créé la table, on lui crée :
- Un controller : Héritant de AppController, avec une variable $scaffold sui se chargera de presque tout et une variable $name avec son nom.
- Un model : Avec la même variable $name et une variable $displayField qui correspond a la designation dans la BDD (ex: "nom");
  Il doit contenir la fonction typeSuiviNameIsUnik($name) {} (qu'on retrouve dans sites ou organisme) ainsi que le $validate comprenant les validations necessaires.
- Une vue, selon l'utilité, n'est pas forcément nécessaire grâce au scaffold.

Par la suite, pour lister son contenu via un lien dans "outils" par exemple, il suffit de créer le chemin dans view/pages/tools.ctp.

10.2. Version du framework CakePhp utilisé dans le projet

Niveau : F
(EP updated 4/2/19)

L'information est dans le fichier **vendor/cakephp/cakephp/VERSION.txt**
Ou bien, exécuter cette ligne php :

```
Configure::version();            // 3.5.11
```

On peut aussi taper la commande : ./VERSION
Elle donne la version de tous les composants du projet

10.3. Les QrCodes

Niveau : M

Le **QrCode** représente l'URL de la vue détaillée d'un matériel
Il est généré à la volée à chaque fois qu'on visualise un matériel en allant sur sa vue détaillée (materiels/view/<materiel_id>).
C'est donc le fichier "vue détaillée" du matériel qui appelle le code de génération du QrCode, soit le fichier src/Template/Materiels/view.ctp
Ce fichier contient le code source suivant qui crée une section html "image" contenant le nom du fichier image PNG du QrCode, généré à la volée par l'action creer() du controleur des QrCodes (src/Controller/QrCodesController.php) :

```
$this->request->getSession()->write("qrUrl", $this->request->env('SERVER_NAME') . $this->request->env('REQUEST_URI'));
$this->requestAction('/QrCodes/creer/');
echo $this->Html->image('qrcodes/' . $this->request->getSession()
->read("filename"), [
'alt' => 'QrCode : ' . $materiel->numero_laboratoire,
```

```
'style' => 'float: right'
]);
```

Voici le code source de la fonction **creer**() du controleur des QrCodes ;

```
use \PHPQRCode\QRcode;
...

// Le fichier QrCode porte simplement le nom de l'ID de la session suivi de l'extension ".png".
// Par exemple, "0fga4e9e6osa97iunfsvk8m8m8.png".
$fileName = $this->request->getSession()->id() . '.png';

// L'image QrCode est créée dans le dossier webroot/img/qrcodes/
$cakephpPath = str_replace('webroot/index.php', '', $_SERVER['SCRIPT_FILENAME']);
$qrCodePath = $cakephpPath . 'webroot/img/qrcodes/' . $fileName;

$this->request->getSession()->write('filename', $fileName);
$this->request->getSession()->write('qrCodePath', $qrCodePath);

// Creation du QrCode avec la methode png() de PHPQRCode\QRcode
if ($message == null) {
        return QRcode::png($this->request->getSession()->read('qrUrl'), $qrCodePath);
} else {
        return QRcode::png($message, $qrCodePath);
}
```

10.4. Contrôleur des pages web “simples” (statiques)

Niveau : F
(updated 18/2/20 - EP)
Voir https://book.cakephp.org/3/fr/controllers/pages-controller.html

*Le squelette d’application officiel de CakePHP est livré avec un controller par défaut **PagesController.php**. C’est un controller simple et optionnel qui permet d’afficher un contenu statique. La page d’accueil que vous voyez juste après l’installation est d’ailleurs générée à l’aide de ce controller et du fichier de vue **src/Template/Pages/home.ctp**.*

*Ex : Si vous écrivez un fichier de vue **src/Template/Pages/a_propos.ctp**, vous pouvez y accéder en utilisant l’url **http://exemple.com/pages/a_propos**. Vous pouvez modifier le controller Pages selon vos besoins.*

*Quand vous « cuisinez » (avec la commande “bake”) une application avec Composer, le controller Pages est créé dans votre dossier **src/Controller/***

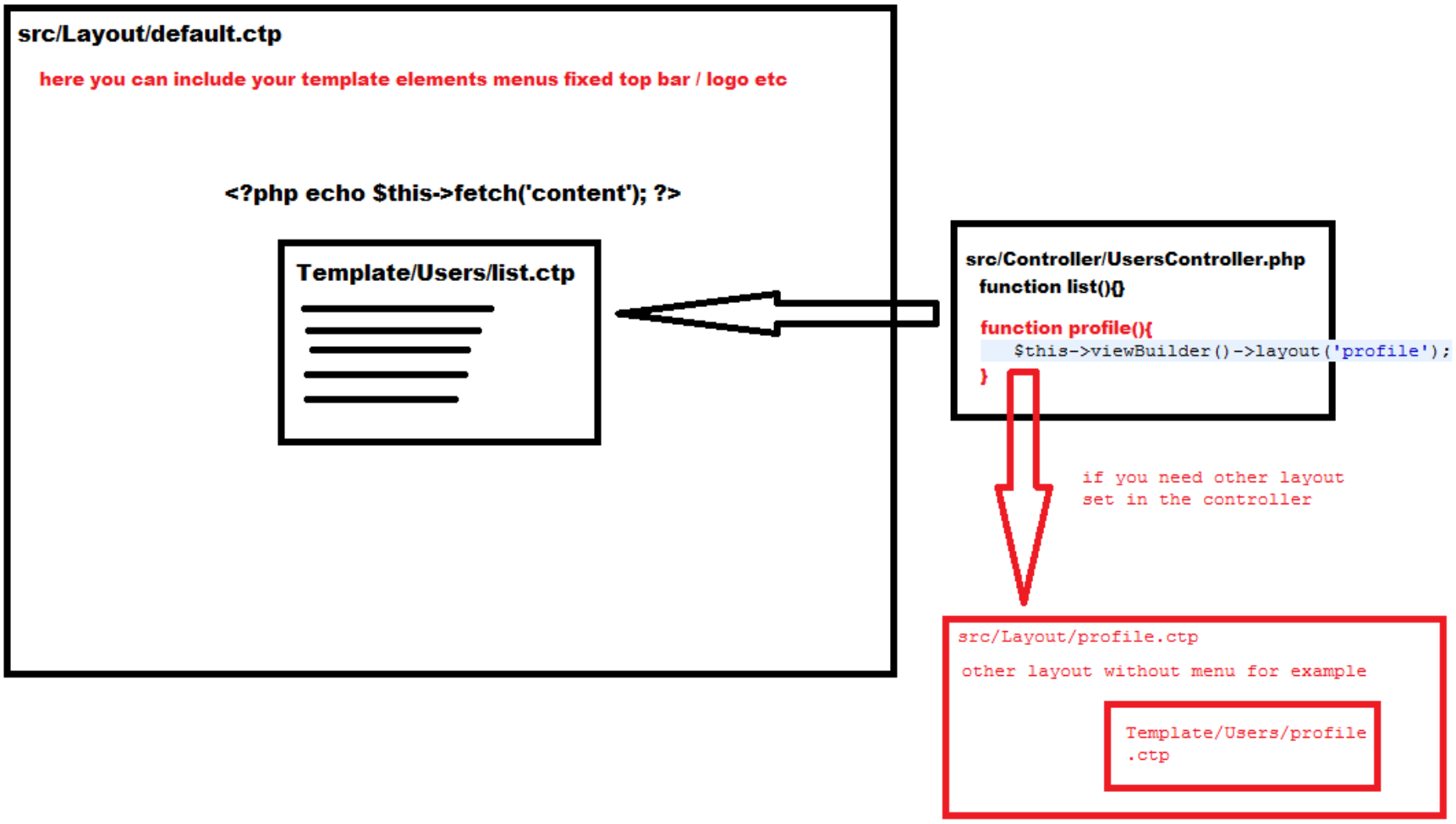
Les pages “simples”, c’est à dire les pages plus ou moins statiques dont le contenu n’est pas dépendant de la BD, dont la structure est souvent assez simple, et qui n’ont pas d’action associée (par exemple des actions telles que “edit”, “add”, “delete”..., du coup c’est l’action par défaut “**display**” qui est appelée) se trouvent toutes dans src**/Template/Pages/**
Exemples :

- Page “à propos” = src/Template/**Pages**/about.ctp
- Page d'accueil = src/Template/**Pages**/home.ctp
- Page “Outils” = src/Template/**Pages**/tools.ctp
- Page des imprimantes = src/Template/**Pages**/printers.ctp

Toutes ces pages ont le même contrôleur qui est src/**Controller/PagesController.php**
Ce contrôleur ne définit qu’une seule action : **display**()

10.5. Structure générale d’une page web (TEMPLATE) = default.ctp

Niveau : M
(updated 18/2/20 - EP)



L'affichage d'une page web suit le workflow suivant (nous prenons ici l'exemple de la liste des matériels) :

1) Affichage de la page **src/Template/Layout/default.ctp**

2) La section "**content**" de default.ctp est remplacée par le contenu de **src/Template/Materiels/index.ctp**

C'est ce que fait la ligne 137 de default.ctp :

<?= $this->**fetch('content')** ?>

3) la page index.ctp **charge les éléments "menu" et "menu_index"**, à partir du dossier **src/Template/Element/**

C'est ce qui est fait à partir de la ligne 454 de index.ctp :

```
<div class="actions">
        <?php
                echo $this->element('menu');
                echo $this->element('menu_index', [
                'pluralHumanName' => 'Matériels',
                'singularHumanName' => 'Matériel'
                ]);
        ?>
</div>
```

La fonction **element**('menu') charge le fichier **src/Template/Element/menu_index.ctp**

La fonction **element**('menu_index') charge le fichier **src/Template/Element/menu_index.ctp**

DONE:
je pense qu'on devrait plutôt faire ces appels à element() **directement dans default.ctp** car la plupart des pages ont besoin des menus...

Voici la structure générale du template général src/Template/Layouts/**default.ctp** :

```
   <div id="container">
        <div id="header">
                LE HEADER AVEC SON LOGO
                <div class="user">
```

```
                BIENVENUE $userName (ou invité)
            </div>
        </div>
        <div id="content">
            <?php echo $this->Session->flash(); ?>
            <?php echo $this->fetch('content'); ?>
        </div>
        <div id="footer">
            LE FOOTER
        </div>
    </div>
```

La DIV "**content**" insère 2 contenus :

- le message flash éventuel (qui dit si une opération demandée s'est bien passée ou pas...)
- la section "**content**", qui n'est autre que le **coeur de la page appelée**

La page d'ACCUEIL n'est qu'un "content" parmi d'autres.
Elle se trouve dans src/Template/Pages/home.ctp
(elle est affichée par le controleur de pages nommé PagesController)

10.6. Ajouter une nouvelle action sur un contrôleur

Niveau : M

Ex: ajout de l'action statusCreated() dans le controleur Materiel

Il faut l'ajouter à 2 endroits dans MaterielsController.php :

a) ajouter une methode statusCreated() :

```
public function statusCreated($id = null, $from = 'index') { ... }
```

b) ajouter un cas 'statusCreated' dans la methode isAuthorized() qui doit retourner true pour autoriser cette action :

```
case 'statusCreated': ... return true; ...
```

La voici en détail :

```
case 'statusCreated': // de-validation d'un materiel (repasse à CREATED)
                        $id = (int) $this->request->getParam('pass.0');
                        // Admin + ok
                        if ($this->USER_IS_ADMIN_AT_LEAST()) return true;
                        // Resp ok if same group
                        if ($this->USER_IS_RESP() && $this->isRespGroup($id, $user[$ldapAuthType])) return true;
                        // User not ok
                        break;
```

10.7. Champ facultatif/obligatoire

Niveau : M

Comment rendre facultatif ou obligatoire un champ, et définir les vérifications à faire sur ce champ ?

=> src/Model/Table/<Model>Table.php
ex : src/Model/Table/MaterielsTable.php pour définir les règles sur les champs de la table "materiels"

10.8. Où définir les éléments passés à une vue (c'est à dire à une action) ?

Niveau : M

=> src/Controller/<Model>Controller/nom_de_la_vue_ou_action()

ex : src/Controller/MaterielsController/edit() pour définir (avec set()) les éléments passés à la vue "edit" des materiels

10.9. Adapter LabInvent au laboratoire utilisateur

Niveau : M
(updated 19/12/18 - EP) ⇒ **A compléter**

Vues (Pages) concernées :
- src/Template/Layout/default.ctp (template)
- src/Template/Pages/home.ctp (Accueil)
- src/Template/Pages/about.ctp (A propos)

- src/Template/Pages/printers (Etiqueteuses)

Controleurs concernées :
- src/Controller/MaterielsController.php (fonction getLabNumber())

Documents concernés :
- Etiquette (**fait**)
- src/Template/Documents/admission.ctp (Document d'admission UPS et CNRS)

10.10. Ajouter une nouvelle page web

Niveau : M
(updated 19/12/18 - EP)

Par exemple, voici ce que j'ai fait pour ajouter la page "about", qui est accessible via l'url https://inventirap.irap.omp.eu/pages/about
1) Aller dans src/Template/Pages/
1) Faire un copier/coller d'une page existante, par exemple infos.ctp, et l'appeler about.ctp
2) Remplir cette page avec le contenu souhaité
3) Tester que cette page est bien accessible via l'url http://.../pages/about
4) Ajouter un lien vers cette page, soit dans le menu outils (src/Template/Pages/tools.ctp), soit dans le menu général (src/Template/Element/menu.ctp)
5) Si cette page ne doit pas être accessible à tout le monde, définir le profil minimum exigé dans src/Controller/PagesController.php, dans la fonction display() :

```
if ($page == about) {
        // Autoriser seulement à partir du role ADMIN
        if (! $this->USER_IS_ADMIN_AT_LEAST()) return $this->redirect('/');
}
```

(ou bien ajouter une ligne dans le tableau $minProfileAllowedForPage)

10.11. Recherche de matériels

Niveau : M
(updated 19/12/18 - EP)

3 methodes :
- Recherche générale (champ “Recherche” sous le menu général à gauche) : src/Template/Element/menu.ctp
    ⇒ appelle l’action “materiels/find”
- VUE de recherche : src/Template/Materiels/find.ctp
- ACTION de recherche : src/Controller/MaterielsController (methode find())

10.12. Autres HOWTO plus anciens

Le contenu de ce chapître est à utiliser avec précaution car, étant assez ancien, il risque de ne pas être complètement utilisable (bien qu'il puisse encore l'être dans la plupart des cas). Cependant il reste utile pour information.

INSTALLATION
****
La procedure d'installation est decrite dans le fichier INSTALLATION.txt qui se trouve dans le dossier install/
Pour une installation à partir d'Eclipse, voir le document install/manual_install/INSTALLATION_MANUELLE_mode_expert.txt

OU EST QUOI (WHERE IS WHAT) ?
****

- OU mettre a jour la VERSION du soft (AVANT CHAQUE COMMIT) ?
En attendant mieux, on fait ça dans le fichier README.md (ça sera automatiquement affiché par src/Template/Layout/default.ctp)

- OU EST LA PAGE GENERALE (qui contient tous les elements) ? : cakephp/app/View/Layouts/default.ctp

- OU EST LA PAGE D'ACCUEIL ? : app/View/Pages/home.ctp

- OU SONT LES MENUS ? : app/View/Elements/
  - menu general + Recherche generale : menu.ctp
  - sous le menu general, et sous le champ "Recherche", titre du modele a ajouter : menu_index.ctp

- OU EST LA FEUILLE DE STYLE CSS ? : cakephp/app/webroot/css/inventirap.css

- OU EST DEFINIE LA PAGINATION ?
Dans le Controleur
Ex : la pagination des materiels est definie dans app/Controller/MaterielsController.php

```
    public $paginate = array(
            'limit' => 50,
            'order' => array('Materiel.id' => 'desc'));
```

- OU SONT LES INFOS CONCERNANT UNE ENTITE quelconque (Materiel, Emprunt, Suivi, Utilisateur) ?
par exemple, ou trouver les infos sur l'entite "Materiel" ? : Aller dans cakephp/app/
- le Modele : Model/Materiel.php
- le Controleur : Controller/MaterielsController.php
- les Vues (templates) : View/Materiels
    - Vue de consultation : scaffold.view.ctp
    - Vue d'ajout (add) et edition (edit) : scaffold.form.ctp
    - vue de liste : index.ctp

- OU SONT DEFINIS LES ROLES (ACL) ?
Dans app/Model/Utilisateur.php :

```
    private $acceptedRoles = array ('Utilisateur', 'Responsable', 'Administration', 'Super Administrateur');
    public function getAuthenticationLevelFromRole($role) {
            if ($role == 'Utilisateur')
                    return 1;
            elseif ($role == 'Responsable')
                    return 2;
            elseif ($role == 'Administration')
                    return 3;
            elseif ($role == 'Super Administrateur')
                    return 4;
            return 0;
    }
```

ANCIEN FICHIER DE CONFIG labinvent.php
****

(updated 19/12/18 - EP)

(cf http://book.cakephp.org/2.0/fr/development/configuration.html#loading-configuration-files)
Ce fichier n’est plus utilisé car maintenant la configuration est mise dans une table “configurations”.
Je laisse quand même cette section pour information sur la manière de gérer la configuration via un fichier php.

1) J'ai créé le nouveau fichier de config dans app/Config/ (par exemple labinvent.php)
Ce fichier doit au minimum contenir un tableau nommé $config :

```
$config = array(
        'labName' => 'IRAP',
        ...
);
```

2) Charger ce nouveau fichier de config au démarrage
Pour cela, ajouter cette ligne dans app/Config/bootstrap.php :

```
Configure::load('labinvent');
```

3) Lire (ou même modifier) les paramètres de ce fichier de config, depuis N'IMPORTE OU (controleur, modèle, vue) :

```
debug(Configure::version()); // pour afficher la version de Cakephp
debug(Configure::read()); // tout lire
debug(Configure::read('localisation')); // le tableau $localisation
$labName = strtoupper(Configure::read('localisation.labNameShort'));
debug(Configure::read('localisation.labName'));
if (Configure::read('USE_LDAP')) ...
if (Configure::read('debug') > 0 ) ...
```

On peut même modifier la valeur d'un paramètre dynamiquement comme ceci :

```
Configure::write('debug',2)
```

C'est d'ailleurs ce qui est fait dans app/Config/core.php

4) Penser à modifier le script d'installation install/installation.sh pour qu'il prenne en compte ce nouveau fichier

GOOGLE ANALYTICS INTEGRATION
****
adapté de http://blog.janjonas.net/2010-01-31/cakephp-google-analytics-integration

1) Copier ce code dans src/Template/Element/google-analytics.ctp :

```
<?php
$gaCode = Configure::read('google-analytics.tracker-code');
if ($gaCode) {
```

```
$googleAnalytics = <<<EOD
<script type="text/javascript">
  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
  (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
  m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
  })(window,document,'script','//www.google-analytics.com/analytics.js','ga');

  ga('create', 'UA-45668893-3', 'omp.eu');
  ga('send', 'pageview');

</script>
EOD;
echo $googleAnalytics;
}
?>

/* ANCIEN CODE JAVASCRIPT :
<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
</script>
<script type="text/javascript">
try {
var pageTracker = _gat._getTracker("$gaCode");
pageTracker._trackPageview();
} catch(err) {}</script>
*/


2) Inclure le view element dans src/Template/Layout/default.ctp (juste avant </body>):
<?php echo $this->element('google-analytics'); ?>

3) Définir le tracker code dans la configuration (/app/Config/core.php):
Configure::write('google-analytics.tracker-code', false); // disables Google Analytics
Configure::write('google-analytics.tracker-code', 'YOUR-TRACKING-CODE'); // enables Google Analytics
```

GENERALITES
****

- URL : http://localhost/inventirapsvn/cakephp/

- BD admin : http://localhost/phpmyadmin

- LOG : cakephp/app/tmp/logs/inventirap.log

- (UPDATE : cd Inventirap/ ; chmod -R 777 ./cakephp/app/tmp/)

- Eclipse config :
Setup syntax highlighting for .ctp files:
Window > Preferences > General > Appearance > content Types > Text > PHP Content type > Add.. , then put in *.ctp.

- Cakephp config : cakephp/app/Config/
Mode debug ON (pour voir les erreurs et pour vider le cache en cas de changement sur la BD)
core.php : Configure::write('debug', 1);

- Fichier de demarrage :
cakephp/index.php (qui pointe sur cakephp/app/webroot/index.php)
(Attention, il y a aussi un fichier cakephp/app/index.php qui pointe sur webroot/index.php)
(ROOT doit pointer sur le dossier cakephp/)
Le fichier execute ensuite est cakephp/lib/Cake/bootstrap.php qui lance ./Core/App.php
and so on...

ACL (Controle d'acces aux ressources)
****

NB: Cette section n'est sans doute plus très à jour ; sur ce sujet, il est préférable de lire le document docs/userguide/ACL.doc (ou .pdf)

Synthese sur les droits selon le profil

Profils (roles), dans le sens du pouvoir croissant :
Qq = Utilisateur Quelconque (lambda)
Rp = Responsable
Ad = Administration
Sa = Superadmin

Actions :
C = Create
R = Read (voir, consulter)
U = Update (mettre a jour)
D = Delete (supprimer)
V = Valider un materiel CREATED --> passe alors en statut VALIDATED
A = Demander l'Archivage d'un materiel VALIDATED --> passe alors en statut TOBEARCHIVED
S = Sortir de l'inventaire (Valider une demande d'archivage d'un materiel TOBEARCHIVED) --> passe alors en statut ARCHIVED
E = Exporter

Par defaut, le superadmin a acces a TOUT

Materiels :
- Qq a les droits C, R (sauf champs admin), U (si createur et sauf champs admin), A, D (si CREATED et owner)
- Rp a les droits C, R (sauf champs admin), U (sauf champs admin), D (si CREATED), V, A, E
- Ad a les droits C, R, U (ssi NOT ARCHIVED), D (si CREATED), V, (A mais inutile car fait directement S sans passer par A), S, E

Suivis et Emprunts :
- Dans tous les cas, on ne doit pas pouvoir emprunter ou suivre un materiel non valide (CREATED)
- Qq a les droits C, R, U (si createur), D (si createur)
- Rp a les droits C, R, U, D
- Ad a les memes droits que Rp

VUES specifiques :
- Acces aux Outils : reserve a Rp et Ad (vue contenant des liens vers differentes ressources telles que utilisateurs, materiels, categories...)
- L'administration a une vue resumee sur la page d'accueil (liens directs vers actions a faire)
- L'administration a une vue enrichie de la liste des materiels :

- filtres (y-compris sur ARCHIVED)
- export en CSV (pour tableur Excel)
- upgrade du statut (validation et sortie)

Autres regles (de gestion) importantes :
- un materiel non valide (CREATED) peut ���tre supprime uniquement par le createur de la fiche, le responsable (Roger), et l'administration
Idem pour la mise ��� jour de cette fiche
- un materiel valide (VALIDATED) n'est pas supprimable
- les champs ADMINISTRATIFS d'un materiel ne sont visibles et modifiables que par l'administration
- par defaut, la liste des materiels affiche tous les materiels SAUF ceux qui sont sortis de l'inventaire (ARCHIVED) qui sont donc masques.
Il est de toutes facons toujours possible (pour l'administration seulement) de les voir, grace au nouveau filtre "Archives" present sur la liste des materiels
- la recherche doit s'effectuer dans TOUS les materiels (y-compris ARCHIVED)

CREER UN CHAMP DATE (VM)
****

Pour expliquer comment créer un champ date fonctionnel, je vais prendre l'exemple de la Date de reception.
Une fois votre colonne créée dans la table materiel, vous devrez faire des modifications dans :

MaterielController : //Passer la date au format français

```
if(isset($this->request->data['Materiel']['date_reception'])){
$this->request->data['Materiel']['date_reception'] = date("d-m-Y", strtotime($this->request->data['Materiel']['date_reception'])); }
```

le Model Materiel : Créer un champ de validation adapté à la date, avec un regex. Et surtout remplir le formatage de date à l'américaine :

```
if (isset($this->data['Materiel']['date_reception'])) {
    $originalDate = $this->data['Materiel']['date_reception'];
    $this->data['Materiel']['date_reception'] = date("Y-m-d", strtotime($originalDate));
```

}

La vue Materiel : - Scaffold.view : Repasser la date en français et l'afficher.
- Scaffold.form : Créer le champ date du formulaire

Puis il faut adapter le champ date reception presque partout ou il y a le champ date acquisition.

COMMENT J'AI FAIT POUR AJOUTER UN 3eme niveau de categorie appele sur_categorie (ou domaine) (EP) ?
****

je me rends compte d'une incoherence de stockage dans la table Materiels.
En effet, quand on saisit un materiel, on doit choisir une categorie ET une sous-categorie.
Du coup, les 2 id (categorie + sous-categorie) sont stockes dans la table Materiels...
Or, c'est inutile car la sous-categorie suffit a determiner la categorie...
Cette redondance pourrait meme amener des incoherences dans la table Materiels si par exemple on fait des modifications
dans les tables categories et sous_categories sans les repercuter dans la table materiels !!
Je suppose que c'est un choix de facilite qui a ete fait par upsilon.

Donc, si vous etes ok, et a moins que Upsilon me dise qu'il y avait une bonne raison de faire ce choix (j'en doute),
je propose de modifier le code pour que seule la sous-categorie soit stockee (on ne stocke plus la categorie).

En plus, cela simplifiera l'ajout du 3eme niveau "sur-categorie" puisque lui-meme n'aura pas besoin d'etre stocke etant donne qu'il est
automatiquement determine par la categorie.
On aura alors :
sous_categorie_id -> categorie_id -> sur_categorie_id
Avec seulement la sous_categorie, on pourra determiner la categorie, et donc la sur_categorie.

Du coup, si je fais cette modif, on pourrait meme se permettre de demarrer officiellement INVENTIRAP rapidement.
En effet, l'ajout de la sur-categorie ne change rien au contenu des tables "administratives" (materiels, emprunts, suivis),
et donc on pourra le faire plus tard, meme apres que l'administration ait commence a remplir la BD.
Ca sera totalement transparent.

1) suppression de la redondance (categorie_id) dans la table materiels

a) impact sur la vue index de Materiel
add/edit/view/index

2) ajout d'une sur-categorie

a) impact sur la BD
- ajout d'une nouvelle table sur_categories(id,nom) :

```
CREATE TABLE IF NOT EXISTS sur_categories (
  id int(11) NOT NULL AUTO_INCREMENT,
  nom varchar(45) DEFAULT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY nom_UNIQUE (nom)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1;
```

- ajout dans la table categories d'une cle etrangere sur_categorie_id :

```
ALTER TABLE categories
    ADD sur_categorie_id INT( 11 ) NULL DEFAULT NULL,
     ADD CONSTRAINT fk_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO ACTION ON
UPDATE NO ACTION;
```

- ajout dans la table materiels d'une cle etrangere sur_categorie_id :

```
ALTER TABLE materiels
    ADD sur_categorie_id INT( 11 ) NOT NULL after designation,
    ADD CONSTRAINT fk_materiels_sur_categorie_id FOREIGN KEY (sur_categorie_id) REFERENCES sur_categories (id) ON DELETE NO
ACTION ON UPDATE NO ACTION;
```

(attention, il faut d'abord vider les lignes des tables suivis, emprunts, et materiels)

```
delete from suivis;
delete from emprunts;
delete from materiels;
```

Il faut aussi ajouter des sur-categories :

```
# Ajout de quelques sur-categories
insert into sur_categories (id,nom) values
(1,'Electronique'),
(2,'Informatique'),
(3,'Instrumentation')
(4, 'Logistique'),
(5, 'Mecanique'),
(6, 'Optique')
;

# Relier toutes les categories a une sur-categorie (la 1)
update categories set sur_categorie_id=1 where id<10;
update categories set sur_categorie_id=2 where id>=10 and id<20;
update categories set sur_categorie_id=3 where id>=20;
```

b) impact sur les modeles (cakephp/app/Model)
- SurCategorie.php : copie de Categorie.php avec "hasMany categorie"
- Categorie.php : + belongsTo="SurCategorie"

c) impact sur les controleurs (cakephp/app/Controller)
- SurCategoriesController.php : minimaliste (comme CategoriesController)
- CategoriesController.php : +getBySurCategorie()

d) impact sur les vues (cakephp/app/View)
- SurCategorie : no view (scaffold ?)
- Categorie : actuellement no view (scaffold ?)
--> creer une vue (comme SousCategorie)
+ get_by_surcategorie.ctp
+ scaffold.form.ctp

e) View/Pages/tools.ctp : ajouter une entree au menu pour les Sur-Categories

f) impact sur TOUTES les vues de Materiel
add/edit/view/index

dans la vue index, remplacer la categorie par la sur-categorie
GROS BOULOT sur la vue ADD/EDIT (+ javascript)

TESTS
****

A - EXECUTION
****

Prérequis : PHPUNIT 3 doit être déjà installé et accessible depuis la console (phpunit -version) via le fichier php.ini
(ATTENTION : cakephp 2.x n'est pas compatible avec PhpUnit 4)

Avec XAMPP (conseillé) : PhpUnit 3 est déjà inclus
Avec MAMP (+ difficile) : pour installer PhpUnit, suivre cette documentation :
http://www.dolinaj.net/software-installation/mac/how-to-install-phpunit-with-mamp-on-mac/

Procédure à suivre pour pouvoir exécuter les tests unitaires et fonctionnels livrés avec le logiciel :

1) Ajouter une nouvelle configuration de base de données dans votre fichier app/Config/database.php pour la BD de test

Exemple de configuration :

```
    public $test = array(
            'datasource' => 'Database/Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'database' => 'test_labinvent',
            'login' => 'root',
            'password' => '',
    );
```

2) Créer la BD de test (vide)
A l'aide de phpmyadmin ou bien avec un client mysql quelconque,
créer une BD de test vide que vous nommerez avec le même nom que celui donné dans la configuration ci-dessus,
soit pour l'exemple, "test_labinvent"
Syntaxe SQL : "create database test_labinvent"

3) Passer en mode "debug"
Dans votre fichier de configuration labinvent.php, mettez votre paramètre "debug" à 1 ou 2 (mais pas à 0) :
```
        'debug' => 2,
```

4) Se connecter à l'application
Les test ne passeront pas si vous n'êtes pas connectés

5) Exécuter les tests

a) Exécution depuis l'application (conseillé) :
ATTENTION : vous devez être logué pour que les tests passent !!!
Il suffit d'aller à l'URL /test.php de votre installation du logiciel LabInvent
(vous pouvez aussi essayer l'URL "/app/webroot/test.php", ou encore "/cakephp/test.php")
Cette page de tests se trouve dans cakephp/app/webroot/
Vous avez alors accès à 2 types de tests :
- App : Tests ==> les tests écris pour tester l'application Labinvent
- Core : Tests ==> les tests fournis avec cakephp pour tester le framework
Pour exécuter tous les tests liés à l'application Labinvent (à faire systématiquement avant de commiter tout changement) :
cliquer sur "Tests" sous "App", puis sur "AllTests"
("AllController" exécute tous les tests de controleurs ; "AllModel" exécute tous les tests de modèles)

b) Execution depuis la console :
Aller dans le repertoire app/
Pour tester le controleur MaterielsController :
./Console/cake test app Controller/MaterielsController

B - ECRITURE DE NOUVEAUX TESTS
****

cf http://book.cakephp.org/2.0/en/development/testing.html

Aller dans app/Test/

Ce dossier contient l'arborescence suivante :

- Case/ : les tests
  - Controller/ : les tests de controleurs
  - Model/ : les tests de modèles
  - View/ (peu ou pas utilisé) : les tests de vues (ou de helpers)
  - AllControllerTest.php : exécution de tous les tests de controleurs
  - AllModelTest.php : exécution de tous les tests de modèles
  - AllTestsTest.php : éxécution de TOUS les tests

- Fixture/ : les différentes initialisations nécessaires dans la BD de test pour pouvoir executer les tests
Ces "fixtures" sont automatiquement executees AU DEBUT de chaque test.
Ce dossier contient un fichier pour chaque table pour laquelle on a besoin d'une "fixture".

1) Les "fixtures"

La façon la plus basique de creer une fixture pour une table donnee est de la realiser automatiquement a partir d'une copie de la table de la vraie BD.
Par exemple, pour que la table "categories" de la BD de test contienne la meme chose que la table de la vraie BD, il suffit de creer un fichier CategorieFixture.php contenant ceci :

```
class CategorieFixture extends CakeTestFixture {
    public $import = array('model' => 'Categorie', 'records' => true);
}
```

Au demarrage des tests, cette table sera chargee automatiquement avec les vraies donnees.
A la fin des tests, cette table sera vidée.

Dans le cas particulier de la table "materiels", on prefere l'initialiser nous-memes avec des valeurs choisies.
Exemple d'une fixture avec 2 materiels (dans le fichier MaterielFixture.php) :

```
class MaterielFixture extends CakeTestFixture {

    public $import = 'Materiel'; // import only structure, no record

    public $records = array(
                array(
                            'designation' => 'matos1',
                            'sur_categorie_id' => 1,
                            'categorie_id' => 11,
                            'materiel_administratif' => 0,
                            'materiel_technique' => 1,
                            'status' => 'CREATED',
                            'nom_createur' => 'Pallier Etienne',
                            'nom_modificateur' => 'Jean Administration',
                            'nom_responsable' => 'Jacques Utilisateur',
                            'email_responsable' => 'Jacques.Utilisateur@irap.omp.eu',
                ),
                array(
                            'designation' => 'matos2',
                            'sur_categorie_id' => 1,
                            'categorie_id' => 11,
                            'materiel_administratif' => 0,
                            'materiel_technique' => 1,
                            'status' => 'CREATED',
                            'nom_createur' => 'Pallier Etienne',
                            'nom_modificateur' => 'Jean Administration',
                            'nom_responsable' => 'Jacques Utilisateur',
                            'email_responsable' => 'Jacques.Utilisateur@irap.omp.eu',
                ),
    );
}
```

2) Les tests

Prenons l'exemple des tests ecrits pour le controleur des materiels (MaterielsController). Il devra s'appeler MaterielsControllerTest et aura la structure suivante :

```
class MaterielsControllerTest extends ControllerTestCase {

    // Liste des fixtures à charger avant l'execution des tests
    public $fixtures = array('app.materiel', 'app.sur_categorie', 'app.categorie', 'app.sous_categorie',
                    'app.groupes_thematique', 'app.groupes_metier', 'app.suivi', 'app.emprunt'
    );

    // Initialisations diverses a faire avant chaque test
    public function setUp() {
        parent::setUp();
    }

    // un 1er test
    public function testMonPremier() {
        $result = $this->testAction(...);
        $this->assert...('resultat attendu', $result);
    }

    // un 2eme test
    public function testMonDeuxieme() {
        $result = $this->testAction(...);
        $this->assert...('resultat attendu', $result);
    }

    ...

}
```

Voir le vrai fichier Test/Case/Controller/MaterielsControllerTest.php

3) Execution

Exemple avec l'execution du test MaterielsControllerTest

a) Execution depuis le site web :
/test.php?case=Controller%2FMaterielsController
Ajouter &debug=1 à l'url pour voir tous les messages de debug

b) Execution depuis la console :
Dans le repertoire app : ./Console/cake test app Controller/MaterielsController
Ajouter --debug pour voir tous les messages de debug

DATE PICKERS
****

Pour fontionner, le datePicker fait appel dans la page "View/Layout/default" à 3 scripts (jquery-1.5.2.js, jquery-1.8.12.js, DatepickerConfig.js) présents dans le repertoire "webroot/js/" et à un fichier "Theme" (smoothness.css) présent dans le repertoire "webroot/css/"

Le thème global peut très facilement être changé en téléchargeant le fichier css de son choix, à cette adresse: "http://jqueryui.com/themeroller/" et en remplaçant celui se trouvant dans "webroot/css/"

Les options du datePicker sont modifiables dans le fichier "webroot/js/DatepickerConfig.js" et sont assez explicites.
Malgré cela, pour plus de précisions, la doc est facilement consultable à cette adresse: "http://jqueryui.com/datepicker/"

11. Cycle de développement à respecter

(11 commandements)

Je veux apporter un changement (correction ou evolution) au projet, comment dois-je faire ?

1) Mettre à jour la version du projet et la date dans le fichier README.md (ça sera automatiquement affiché par src/Template/Layout/default.ctp)

2) Selectionner le changement a apporter (une demande) dans le Redmine du projet (https://projects.irap.omp.eu/projects/inventirap) :
- soit depuis la liste des demandes : onglet Demandes
- soit depuis la roadmap : onglet Roadmap, cocher la case "Anomalie", cliquer sur Appliquer, puis "version 1.3" (la version en cours depuis fin 2012)
Commencer de préférence par les "anomalies" parmi celles qui ont la plus haute priorité (et faire les "évolutions" dans un 2ème temps).
Choisir une demande et cliquer dessus pour aller sur sa fiche detaillee et voir le travail a faire.
Cliquer sur "mettre a jour", selectionner le statut "En cours", modifier eventuellement d'autres champs..., puis cliquer sur "Soumettre".
Noter l'URL de cette fiche (par exemple : https://projects.irap.omp.eu/issues/1050)
NB : Si la demande n'existe pas encore, la creer :
- cliquer sur l'onglet "Nouvelle demande"
- Selectionner "Anomalie" ou "Evolution"
- Positionner le statut a "En cours" si on veut travailler aussitot dessus (sinon, laisser a "Nouveau")
- Completer le reste de la fiche de demande (mettre "Assigne a" a "<<moi>>", choisir la version cible "version 1.3" ou "version 1.4", ...)
- cliquer sur le bouton "Creer"

3) Verifier que cette nouvelle demande apparait bien dans la roadmap (cliquer sur onglet "Roadmap", ..., puis version 1.3)

ÉTAPE OPTIONNELLE MAIS FORTEMENT CONSEILLÉE :
4) Ecrire un test qui vérifie que cette fonctionnalité ne marche pas encore (bug) ou bien n'est pas encore implémentée
On utilise ici l'approche TDD (Test Driven Development)
Ce test ne devrait pas passer (il est au rouge) ; il passera plus tard, quand on aura écrit le code nécessaire.
Bien sur, c'est dans la mesure du possible, car on ne peut pas TOUT tester.
Dans tous les cas, il faut ecrire un test qui s'approche le plus possible de la réalité à tester.

Voir pour cela la section "TESTS" (à la fin de ce document)

5) Faire les changements necessaires dans le code Php
Si ce changement implique aussi un changement dans la base de donnees,
copier le script SQL correspondant à ce changement dans un fichier database/update/db-update-YYYY-MM-DD.sql
portant la date du jour (voir des exemples dans database/update/old/).
Plus tard, il faudra penser à intégrer ce changement dans le script general de creation de la BDD database/BDD_IRAP.sql
(et/ou éventuellement les fichiers Insert_TablesFunct.sql, Insert_Users.sql, et Upd_TableConstraints.sql)
et donc deplacer le script de modification database/update/db-update-YYYY-MM-DD.sql dans database/update/old/db-update-YYYY-MM-DD.sql

6) Tester manuellement ce changement jusqu'a ce qu'il soit totalement OK
a) faire quelques tests manuels
b) Le test écrit à l'étape (4) doit maintenant passer (il est au vert).
Il doit être inclus dans l'ensemble des tests accessibles par le lien "AllTests".
c) Test de non régression : afin de s'assurer que cette modification du code n'entraîne aucune régression sur le reste du code,
tous les autres tests écrits avant doivent aussi passer (vert) : pour cela, exécuter l'url /test.php?case=AllTests

7) Compléter le fichier /README.txt, section HISTORIQUE DES VERSIONS (fichier situé à la racine du projet)
Y mettre le même commentaire que ce que tu mettras lors du commit

8) Mettre a jour TON code (en mode console, "svn update" depuis la racine du projet, ou bien depuis Eclipse, clic droit sur le projet, "Team/Update")
C'est important, au cas ou quelqu'un d'autre aurait fait des modifs (avant ou en meme temps que toi), et pour etre bien sur d'avoir la derniere version

9) Faire un "commit" du code, en collant dans le commentaire l'URL de la demande realisee (exemple : https://projects.irap.omp.eu/issues/1050)

10) Fermer la demande sur redmine
Sur la fiche detaillee, cliquer sur "Mettre a jour", changer le statut a "ferme", changer "% realise" a 100%, (copier l'URL de la fiche), cliquer sur Soumettre

11) Si c'est un changement important, le répercuter sur le site officiel
Le changement est important si c'est une anomalie ou bien si c'est une évolution attendue.
Demander alors au service informatique de le répercuter sur l'installation officielle (faire un "svn update", mail à loic.jahan@irap.omp.eu).
Si ce changement implique aussi la base de données, donner la directive que le fichier database/update/db-update-YYYY-MM-DD.sql doit être exécuté sur leur BDD.
Si ce changement impliquer une modification du fichier de configuration labinvent.php, donner la procédure à suivre dans le mail.